

WINNING GAMES
on the
COMMODORE 64

T. P. Barrett and S.W. Colwill



Ellis Horwood
BOOKS IN COMPUTING

WINNING GAMES ON THE COMMODORE 64

TERRY BARRETT, Freelance Programmer, and
STEPHEN COLWILL, Head of Computer Studies,
Sandhurst School, Surrey

This book takes the reader from a simple understanding of BASIC and builds up in single stages the techniques of graphics programming on the Commodore 64. The CBM is a recent machine on which little other literature in the area of games programming exists.

The book contains many advanced tips and hints which will help the reader to write fast-moving graphics games in BASIC: to facilitate this end, the book includes Basic Loaders for Machine Code routines. One of these routines enables the program to use the joystick quickly and hence speed up the action.

There are 20 high-level arcade-type games, each with a short explanation of the construction of the program, and how to play the game. One program in particular is dissected and analysed in relation to earlier chapters. The book will, with its unusual combination of a teaching approach and numerous games listings, find a wide range of readers.

Readership: Owners and prospective owners of Commodore 64 machines interested in playing arcade-type graphics games, or games of their own construction who wish to be guided through this process at a simple level. Also owners of Commodore 64 machines with a higher degree of programming skill who wish to pick up hints and techniques in the graphics games area.

WINNING GAMES
on the
COMMODORE 64



ELLIS HORWOOD BOOKS IN COMPUTING

Series Editor: A. J. JONES, Brunel University

MASTERING THE COMMODORE 64

A.J. JONES, Brunel University, and G. CARPENTER, Racal Tacticom Ltd.

STRUCTURED PROGRAMMING WITH BBC BASIC

ROY ATHERTON, Bulmershe College, Reading

MASTERING THE VIC-20

A.J. JONES, Brunel University, E.A. COLEY, Maystar Microcomputers Ltd. Wokingham,
and D. G. J. COLE, British Aerospace, Bracknell

BEGINNING micro-PROLOG 2nd Revised Edition

RICHARD ENNALS, University of London

GETTING STARTED ON THE ORIC-1

D.G.J. COLE, British Aerospace, Bracknell

BEGINNING COMAL

BORGE CHRISTENSEN, Tonder College of Higher Education, Tonder, Denmark

WINNING GAMES ON THE COMMODORE 64

TERRY BARRETT, Freelance Programmer, and STEPHEN COLWILL, Sandhurst School, Berkshire

LOGO: A GUIDE TO LEARNING THROUGH PROGRAMMING

PETER GOODYEAR, Avery Hill College, London

WINNING GAMES ON THE VIC-20

TERRY BARRETT, Freelance Programmer, and A.J. JONES, Brunel University

MASTERING THE ACORN ELECTRON

SIMON and JOHN MATTHEWS, University of Exeter

MASTERING THE ZX SPECTRUM

LAWRIE MOORE, freelance writer and lecturer, formerly University of London

BEGINNING LISP

AJIT NARAYANAN, Exeter University

WINNING GAMES on the **COMMODORE 64**

T. P. Barrett and S.W. Colwill



ELLIS HORWOOD LIMITED
Publishers · Chichester

JOHN WILEY & SONS INC
New York · Chichester · Brisbane · Toronto · Singapore

First published in 1984 by

ELLIS HORWOOD LIMITED

Market Cross House, Cooper Street, Chichester,
West Sussex PO19 1EB, England

Distributors:

Australia, New Zealand, South-east Asia:

Jacaranda-Wiley Ltd., Jacaranda Press
JOHN WILEY & SONS INC.
GPO Box 859, Brisbane, Queensland 4001, Australia

Canada:

JOHN WILEY & SONS CANADA LIMITED
22 Worcester Road, Rexdale, Ontario, Canada

Europe, Africa:

JOHN WILEY & SONS LIMITED
Baffins Lane, Chichester, West Sussex, England

North and South America and the rest of the world:

JOHN WILEY & SONS INC.
605 Third Avenue, New York, NY 10158, USA

© 1984 T.P. Barrett and S.W. Colwill/Ellis Horwood Limited

British Library Cataloguing in Publication Data

Barrett, T.P.

Winning games on the Commodore 64.

(Ellis Horwood books in computing)

1. Electronic games 2. Commodore 64 (computer)

3. BASIC (computer program language)

I. Title II. Colwill, S.W.

7948.028'5404 GV1469.2

ISBN 0-85312-695-X (Ellis Horwood Limited)

ISBN 0-471-80725-7 (John Wiley & Sons Inc.)

Typeset by Ellis Horwood Limited

Printed in Great Britain by R.J. Acford, Chichester.

All Rights Reserved

No part of this book may be reproduced or transmitted by any means without the prior permission of the publisher. The only exceptions are for the purposes of review, or as provided for by the Copyright (Photocopying) Act or in order to enter the programs herein onto a computer for the sole use of the purchaser of this book.

Contents

FOREWORD	9
DISK DIRECTORY	10
Section 1	
1 BASIC SKILLS	11
Loops	11
GOTO and subroutines	13
Structuring your programs	14
2 PEEK AND POKE	16
Organization of memory	16
PEEK and POKE	16
Screen and color memory	17
What else is useful?	18
Using what we have learned	18
Alioids program	20
Program structure	28
3 PRINTING TO THE SCREEN	29
PRINT and punctuation	29
Positioning output	30
(1) The TAB command	30
(2) The cursor controls	30
(3) Color	32
(4) Commodore character set	33
(5) Reverse mode	33

4 THE LOGICAL OPERATORS AND, OR AND NOT	35
The decimal system	35
The binary system	36
Converting from binary to decimal	36
Converting from decimal to binary	37
Representing numbers used by the logical operators	37
The operator AND	38
The operator OR	39
The operator NOT	40
Further uses of AND and OR	41
 5 USER-DEFINED GRAPHICS	 43
The Supershifter program	43
Designing your own characters	45
Multi-color mode	47
 6 SPRITES	 51
Designing a sprite	51
Writing a simple sprite program	54
Memory pointers	57
Coloring sprites	58
Positioning sprites	63
Other sprite functions	67
Using AND and OR to isolate bits	67
Priority	69
Collisions, Collisions	70
(1) Sprite/sprite collisions	70
(2) Sprite/background collisions	70
Aeroplane program	70
Table of VIC chip location functions	75
 7 USING THE JOYSTICK	 77
Introduction	77
A BASIC joystick subroutine	79
A machine-code joystick subroutine	82

8 BEGINNER'S SOUND ADVICE	88
Introduction	88
Example programs	93

9 ADVANCED TIPS AND HINTS	95
Some do's and don'ts	95
Some interesting POKEs etc.	96
Further exciting snippets	97
The collision registers	97
The use of the return key in editing	97
Trouble-shooting	98
Utility and demonstration programs	98
Sprite generator	98

Section 2 GAMES LISTINGS	107
Demon Dozer	108
3D Tank Assault	113
Sub Hunt	118
Dog Fight	124
Spider Hunt	130
Dippy Dappy	136
Fun Match	142
Guess My Number	144
Bug Bover	146
Attack of the Tomato	151
Paper, Rock, Scissors	161
Searchlight	164
Flip Flap	176
Laser Spell	180
Track Racer	184
Mine Field	188
Look Out!	197
Drum Kit	201
Math Encounters	205
Cosmic Dodge	210
Galactic Battle	215

Foreword

We have written this book with three types of Commodore 64 users in mind. If you have little or no programming experience, but enjoy playing computer games, turn to Section 2. Here you will find the games programs listings. If you have some knowledge of BASIC and wish to begin programming your own games on the '64 then you will find that Section 1 takes you in easy stages through the intricacies of designing your own characters, creating sprites, and using the joysticks to control the action. If you have already reached a high standard then we hope that you will pick up hints and tips from the final part of Chapter 9 and from the program listings themselves. These are also machine-code routines for moving the character set from ROM to RAM and for the joystick controls. These take the form of 'BASIC LOADERS' so you can use them directly in your BASIC programs. Whatever your level of skill and experience we hope that there is something for everyone! As some of the programs contained in this book contain machine code routines, it is necessary before loading a different program to type in SYS 64738 or turn off the machine and start again.

As far as possible, we have avoided using those nasty Commodore control characters that can often be seen in program listings for the '64 and VIC-20. This should mean that typing in the listings given in Section 2 will be a little less frustrating. If you find that your fingers do seize up in the attempt then remember that all the programs listed in this book are also available on cassette and disk, at a snip, from the publishers.

The Commodore 64 has many possibilities for the would-be BASIC games programmer. So, don't just sit there . . . get on with it!

Acknowledgements

We would like to thank all the people who helped us in putting this book together, particularly Antonia Jones, whose knowledgeable guidance made everything possible. Steve would also like to thank Christine for all her domestic efforts and his parents for their constant support. Liz Coley also richly deserves our thanks for introducing us.

STEVE COLWILL
TERRY BARRETT
Wokingham, Berkshire, England
October 1983

Disk directory

1	TIMER	PRG	18	DIPPY DAPPY	PRG
1	TIMER AGAIN	PRG	3	FUN MATCH	PRG
2	RAINBOW	PRG	2	GUESS MY NUMBER	PRG
2	NEGATIVE	PRG	11	BUG BOVVER	PRG
3	SPRITE DEMO #1	PRG	32	ATTACK OF THE TOMATO	PRG
4	MULT SPRITE DEMO #1	PRG	5	PAPER, ROCK, SCISSORS	PRG
4	FLAGPOLE DEMO	PRG	38	SEARCHLIGHT	PRG
14	AEROPLANE	PRG	10	FLIP FLAP	PRG
4	BASIC JOY	PRG	7	LASER SPELL	PRG
11	M/C JOY DEMO	PRG	9	TRACK RACER	PRG
8	ALIOIDES	PRG	26	MINE FIELD	PRG
5	SUPERSHIFTER	PRG	7	LOOK OUT!	PRG
8	M/C JOYSTICK	PRG	7	DRUM KIT	PRG
12	DEMON DOZER	PRG	12	MATH ENCOUNTERS	PRG
16	3D TANK ASSAULT	PRG	15	COSMIC DODGE	PRG
16	SUB HUNT	PRG	15	GALACTIC BATTLE	PRG
17	DOGFIGHT	PRG	12	BOTH LOADERS	PRG
18	SPIDER HUNT	PRG	289	BLOCKS FREE.	

Cassette tapes or floppy disks containing these programs are available from:

- USA — John Wiley & Sons Inc., 605 Third Avenue, New York, NY 10158, USA.
- UK/Europe — Ellis Horwood Limited, Market Cross House, Cooper Street, Chichester, West Sussex, PO19 1EB.

SECTION 1



Basic skills

In this chapter we intend to run over some of the ground rules of good programming practice. Following these rules will enable the reader to write well structured, readable programs. Of course these rules are not obligatory, but neither is writing 'good' programs.

LOOPS

Most of you will be familiar with the FOR ... NEXT loop. It allows us to repeat a group of instructions a set number of times. For example

```
10 PRINT"THIS IS A";  
20 FOR I=1TO10  
30 PRINT"BORING "  
40 NEXT  
50 PRINT"PROGRAM"  
60 END
```

If you run this program you will see that the word BORING is PRINTed 10 times. This is because of the loop which starts at line 20.

```
20 FOR I=1TO10  
:  
:  
40 NEXT
```

The computer starts at line 20 by assigning a value of 1 to the variable I. The following instructions are executed and then the computer comes to line 40. The NEXT instruction tells the computer to go back to line 20 and assign the new value 2 to the variable I; the following instructions are again executed; the computer again reaches line 48 which tells it to return to line 20, assign a value of 3 to I and so on.

Eventually I will take the value 10 and the loop will be executed for the last time. The computer will then move on to execute the instruction at line 50.

The value of I in the FOR ... NEXT loop can be used inside the loop.

```
10 FORI=1TO10
20 PRINT I
30 NEXT
40 PRINT"THIS IS THE END OF THE LOOP"
50 END
```

This program will print out the numbers from 1 to 10 followed by a message to tell you that the loop is finished.

It is easy to modify the program to print out the even numbers. All we need to do is to tell the computer to start with I as 2 and then to increase I in STEPs of 2. If we change line 10 as follows:

```
10 FORI=2TO10STEP2
```

this will accomplish the task.

We can also count in reverse. Change line 10 to read as follows:

```
10 FORI=10TO1STEP-1
```

Or we can list the even numbers as before but in reverse order by changing line 10 to:

```
10 FORI=10TO1STEP-2
```

We can even have one loop inside another, as in this example:

```
10 FORI=1TO10
20 FORJ=1TO3
30 PRINTI,J
40 NEXTJ
50 NEXTI
60 END
```

This program will simply print out pairs of numbers, starting with

I	J	
1	1	
1	2	
1	3	
2	1	
2	2	etc.

Notice that the 'J' loop is totally contained inside the 'I' loop. The loops are said to be 'nested'. An example of the incorrect nesting of loops is given below.

```

10 FORI=1TO10
20 FORJ=1TO3
30 PRINTI,J
40 NEXTI
50 NEXTJ
60 END

```

At first glance nothing much seems to have changed, but on close inspection you will see that lines 40 and 50 have been swapped. The 'J' loop is now no longer wholly contained inside the 'I' loop. This will confuse the computer and an error message will be generated. To avoid incorrect nesting it is possible to drop the 'I' part in NEXTI and the 'J' part of NEXTJ, shortening them to NEXT. Now the computer will assume that the loops are correctly nested.

GOTO AND SUBROUTINES

The GOTO statement is a very powerful means of directing program flow. It is also the single most misused command in BASIC! GOTO should be used sparingly and thoughtfully. Here is an example of how not to use it.

```

10 INPUTA$
20 IFA$="YES"THEN GOTO40
30 GOTO10
40 PRINT"WHAT A CARRY ON"
50 END

```

Compare this with the program below which does the same thing.

```
10 INPUT A$
20 IF A$ <> "YES" THEN GOTO 10
30 PRINT "MUCH BETTER"
40 END
```

Do not use GOTOs if you can avoid them. This may not always be possible but you can try!

Subroutines are rather cleverer than GOTOs because they can remember where they came from. A subroutine is a collection of statements that need to be executed at several stages of the main program. They are placed at the end of the main program, after the END instruction. To call a subroutine from a program we use the command GOSUB followed by the line number at which the subroutine starts. At the end of the subroutine the command RETURN returns the computer to the main program to the same point at which it left. Here is a simple example of a subroutine in action.

```
10 PRINT
10 PRINT "HI THERE"
20 GOSUB 1000
30 PRINT "HELLO AGAIN"
40 GOSUB 1000
50 PRINT "WE CAN'T GO ON MEETING LIKE THIS"
60 GOSUB 1000
70 PRINT "I REALLY MUST GO NOW"
80 END
1000 REM THIS IS THE SUBROUTINE
1010 PRINT "HELLO I'M THE SUBROUTINE"
1020 RETURN
```

STRUCTURING YOUR PROGRAMS

The ubiquitous GOTO raises its ugly head again! One of the golden rules of good programming practice is to ensure that, as far as possible, each section of your program has ONE entry point and ONE exit point. Following this principle will not only make your programs easier to read and debug, but will also keep your computer happy!

When the computer executes a GOSUB the line number and the position in that line are saved in the 'stack'. The stack is a special area of memory inside the computer. Each time a GOSUB is executed the stack requires 5 bytes in which to store the information about GOSUB's position. This 5 bytes of memory is not cleared until RETURN is reached. Hence if you skip out of a subroutine before reaching the RETURN statement then the stack will not be cleared. Obviously you can only do this a limited number of times before all the available memory is used up. So don't write subroutines constructed like this one:

```

1000 REM THIS IS A SUBROUTINE
1010 ...
1020 ...
1030 IFX>30THEN GOTO200
1040 ...
1050 ...
1060 RETURN

```

We can see that if the condition $X > 30$ is true then the computer will never reach line 1060 and clear the stack.

It may mean being a bit more long-winded in your program construction to achieve this but it will ensure that your programs stay out of trouble.

The same rule should be applied to FOR ... NEXT loops. Always ensure that each and every NEXT in the loop is executed. In other words don't do this:

```

10 FORI=1TO10
20 ...
30 ...
40 IFX>120THEN GOTO200
50 ...
60 ...
70 NEXT

```

If you follow these rules it may save you some headaches later on!



PEEK and POKE

ORGANIZATION OF MEMORY

The Commodore 64's memory is divided up into small blocks which can be thought of as a series of pigeon-holes. To distinguish one pigeon-hole from another each block is given a number. This number is called the address. The proper name for a pigeon-hole is 'location'.

So, the computer's memory is divided into locations. Each location has an address. So, what else is new?

As there are very few statements in BASIC to help us with graphics, we have to delve into the computer's memory. This is done by using two BASIC commands, PEEK and POKE.

PEEK AND POKE

PEEK is a command that looks at a specified location and finds out what number is in there (remember pigeon-holes?). Clear the screen of your machine using the SHIFT and CLR/HOME keys together. Now type in this line:

```
PRINT PEEK(2023)
```

You should find that this prints up the number 32. What's going on?

The number 2023 is the address of a particular location. The location in question is the pigeon-hole that holds information about the bottom right-hand corner of the screen.

The line we typed in tells the computer to PEEK at the location and to print what it finds.

The number 32 is the POKE code for a space. This is because there is nothing printed there. Try typing in this line:

```
PRINT PEEK(56295)
```

This line should return the number 6. In fact the location with address 56295 is the one that holds information about the color of the bottom right-hand corner of the screen.

Clear the screen and type:

```
POKE 2023,1
```

Nothing appears to have happened; but it has. Without clearing the screen type:

```
POKE 56295,0
```

If all is well, then a black letter 'A' will be nestling in the bottom corner of the screen. How did such a miracle come about?

The first POKE instruction pushes the number 1 into location 2023. The number 1 is the POKE code of letter 'A'. Nothing appeared to happen after this POKE because, although the letter 'A' was there, it was colored blue, the same color as the background. The second POKE was necessary to give the 'A' a new color, the number 0 being the color code for black.

Let's recap.

Each position on the screen has two locations associated with it. One of these locations holds information about the character in that position: the other holds information about the character's color.

The Commodore 64's screen is made up of 25 rows, with 40 character spaces in each row. This means that there are 1000 possible positions for characters on the screen.

SCREEN AND COLOR MEMORY

The locations for screen memory run from 1024 in the top left-hand corner, to 2023 in the bottom right. Remember that it is the screen memory which holds the character information.

The corresponding locations for color memory start at 55296 and run through to 56295.

When POKEing a character to the screen you need to POKE to, both the screen memory and the color memory or you will not be able to see the character you have POKEd. A useful relation between the two is as follows:

$$\text{COLOUR MEM LOCATION} = (\text{SCREEN MEM LOCATION}) + 54272$$

To illustrate its use try typing in this program:

```

10 PRINT CHR$(147): REM CLEARS SCREEN
20 SC=1504
30 FOR I=0 TO 39
40 POKE SC+I,26
50 POKE SC+54272+I,2
60 NEXT I
70 END

```

Line 20 sets the initial value of the screen code to one which is approximately half-way down the screen on the left-hand margin.

The loop from line 30 to line 60 POKES locations 1504 to 1543 with the number 26, this being the POKE code for the letter 'Z'. At line 50 the corresponding location in color memory is worked out and POKEd with a 2, this being the color code for red.

If you wanted the row of 'Z's to appear on the bottom line, what value would you give to SC in line 20?

If you wanted two rows of 'Z's, how would you alter line 30?

If you wanted green 'Z's, how would you change line 50?

Write a program to draw alternating rows of yellow 'E's. (The POKE code for 'E' is 5.)

A complete list of POKE codes appears in Appendix E of your User Manual.

Diagrams showing screen and color memory locations, along with the 15 color codes, appear in Appendix G of your User Manual.

WHAT ELSE IS USEFUL?

If you wish to change the background color, this can be achieved by the following POKE:

```
POKE 53281,C
```

where C is one of the color codes given in Appendix G.

The border color may be changed by POKEing 53280 with any of the color codes.

USING WHAT WE HAVE LEARNED

Let's now try to design a program that will give us 'animation' – the first step to programming graphics games.

We want to move a red 'X' across the screen for 3 rows, and will start by printing our first red 'X' at location 1504.

```
10 PRINTCHR$(147):REM CLEAR SCREEN
20 SC=1504
30 POKESC,24:REM "X"
40 POKESC+54272,2:REM COLOUR RED
```

Clear the screen and continue typing:

```
50 FORI=0TO118
60 POKESC+I,32:REM SPACE
70 POKESC+I+1,24:REM "X"
80 POKESC+I+54272,6:REM COLOUR BLUE
90 POKESC+I+54272+1,2:REM COLOUR RED
100 NEXT
110 END
```

Look carefully at what the loop in lines 50–100 is doing. It is not simply POKEing one character to the screen, but two, back-to-back.

The loop is printing ' X'. That SPACE in front of the 'X' is quite deliberate. As the loop is executed, the 'X' moves one place to the right. The SPACE prints over the old 'X', blanking it out. The new 'X' is then printed one place to the right of the old one.

If you try to run this now, you will find that the 'X' goes across the screen like a bat out of hell. To slow things down a little we can place a 'dummy' loop inside the main loop. A 'dummy' loop is one which serves no purpose other than to slow the main loop down. Type in:

```
55 FORJ=1TO500
56 NEXT
```

Now try RUNning again to see the difference. Playing about with the upper value of J will alter the speed of movement across the screen. There is no lower limit to the speed but the upper limit is determined by the length of time required to execute a cycle of the main loop.

There is an important principle involved here. That is, the speed of animation is limited by how long it takes the computer to work through the main program loop. But more of this later!

ALIIDS PROGRAM

So far we have only looked at short routines. This is a longer program designed to demonstrate some of the principles of graphic animation. It is based on the classic 'alien invaders' theme, although for our purposes it is somewhat simplified.

Use the letter 'Z' to move the laser base left. 'X' moves the laser base right. Use the 'M' key to shoot down the invading Alioids.

This program is more written to be easily understood, than provide an exciting game. Together with the program listing, a detailed, line-by-line explanation is included.

```

1 REM  *****
2 REM  *****
3 REM  **                **
4 REM  ** ALIIDS **
5 REM  ** DEMO **
6 REM  ** PROGRAM **
7 REM  **                **
8 REM  *****
9 REM  *****
10 PRINTCHR$(147)
20 POKE53281,0
30 S=1024:A=1999:C=55296:SC=1024:CC=55296
40 FORL=1TO2
50 FORI=1TO10
60 POKE53,112:POKE53+1,87:POKE53+2,87
70 POKE53+3,110:POKE53+41,75
80 POKE53+42,74
90 FORJ=0TO3
100 POKECC+J,5:NEXT
110 POKECC+41,5:POKECC+42,5
120 SC=SC+4:CC=CC+4
130 NEXT
140 SC=S+80:CC=C+80
150 NEXT
200 REM-----MAIN PROGRAM LOOP-----
210 REM---RUBOUT OLD ALIEN---
220 FORI=0TO3

```

```

230 POKES+I,32
240 NEXT
250 POKES+41,32:POKES+42,32
260 GOSUB1000:REM LASER BASE ROUTINE
270 REM---PRINT NEW ALIEN---
280 SC=S+160:CC=SC+54272
290 POKESC,112:POKESC+1,87:POKESC+2,87
300 POKESC+3,110:POKESC+41,75::POKESC+42,74
310 FORJ=0TO3
320 POKECC+J,5:NEXT
330 POKECC+41,5:POKECC+42,5
340 GOSUB1000:REM LASER BASE ROUTINE
350 GETJ$:IFJ#<>" "THEN350
360 S=S+4
370 T=(S-984)/40:IFT=INT(T)THENS=S+40
500 IFPEEK(1744)=32THENGOTO200:REM REDO LOOP
600 REM----END OF GAME----
610 REM---FLASH THE SCREEN---
620 PRINTCHR$(147)
630 FORI=1TO5
640 POKE53281,5:FORJ=1TO300:NEXT
650 POKE53281,2:FORJ=1TO300:NEXT
660 NEXT
670 REM---PRINT MESSAGE---
680 FORI=1TO12:PRINT:NEXT
690 PRINTTAB(11)"OVERUN BY ALIIDS"
700 PRINT:PRINT
710 PRINTTAB(6)"<<PRESS SPACE BAR TO PLAY>>"
720 GETJ$:IFJ#<>" "THEN720
730 GETB$:IFB#<>" "THEN730
740 RUN
1000 REM----LASER BASE SUBROUTINE----
1010 B=54272+A
1020 POKEA,160:POKEA+1,160:POKEA-40,100
1030 POKEA-39,123
1040 POKEB,2:POKEB+1,2:POKEB-40,2

```

Listing continued next page

```

1050 POKEB-39,2
1060 POKEA-2,32:POKEA-1,32:POKEA-42,32
1070 POKEA-41,32:POKEA+2,32:POKEA+3,32
1080 POKEA-38,32:POKEA-37,32
1090 REM---MOVE LASER BASE---
1100 GETA$
1110 IFA$="Z"THENA=A-2:IFA<1984THENA=1984
1120 IFA$="X"THENA=A+2:IFA>2022THENA=2022
1130 IFA$="M"THENGOSUB2000
1140 RETURN
2000 REM-----FIRE SUBROUTINE-----
2010 P=A-119
2020 FORI=1TO22
2030 POKEP,117:POKEP+54272,1
2040 POKEP+40,32
2050 P=P-40
2060 IFPEEK(P)<>32THENGOSUB3000
2070 NEXT
2080 POKEP+40,32
2090 RETURN
3000 REM-----HIT SUBROUTINE-----
3010 POKEP+40,32
3020 IFPEEK(P)=112THENH=P+40
3030 IFPEEK(P)=110THENH=P+37
3040 IFPEEK(P)=75THENH=P-1
3050 IFPEEK(P)=74THENH=P-2
3060 FORI=0TO3
3070 POKEH+I,32:NEXT
3080 FORI=40TO37STEP-1
3090 POKEH-I,32:NEXT
3100 RETURN

```

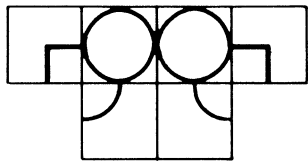
(1) Screen set up

LINE 10: Clears the screen.

LINE 20: Colors the screen black.

LINE 30: Defines variables that will be used later on.
S is the start of screen memory.
SC is the screen code of the current alien character being printed.
A is the screen code of the laser base.
C is the start of color memory.
CC is the color code of the current alien character being printed.

The loop from
LINE 50 to 130: Prints a line of aliens.
LINES 60–110: POKE an alien character to the screen.
The alien is made up of six character blocks.



If the top left character is given a screen code SC then the other blocks that make up the alien are as follows:

SC	SC+1	SC+2	SC+3
	SC+41	SC+42	

The POKE codes for each character are:

112	87	87	110
	75	74	

The color number POKEd in is 5; i.e. green.

- LINE 120: After printing the first alien the screen code is increased by 4, in preparation for the next time through the loop.
- LINE 150: After printing the first row of aliens, the screen code must be increased by 40. This means that a line is skipped before the next row of aliens is printed.

(2) Main program loop

(a) *Rub out old alien*

- LINES 220-250: Remember that S was the original start of the screen and was not altered, during the screen set up, from its original value of 1024. The program now uses S to position the first alien rubout. 32 is the POKE code of a SPACE.
- LINE 260: The subroutine at 1000 is executed at this point to move the laser base.

(b) *Print new alien*

- LINE 280: SC now takes on a value 4 rows down the screen from the position of S. The color code CC is related to the screen code.
- LINES 290-330: Using SC for position, a new alien is colored and printed in the same way as in lines 80-120.
- LINE 340: Laser base subroutine again executed.
- LINE 350: Clears out the keyboard buffer.
To avoid losing characters if the keyboard is pressed several times in quick succession, the extra characters are stored in a 'buffer', until the computer is ready to read them. The buffer can hold up to 10 characters and they are read on a 'first-in-first-out' basis. To ensure that only one character is available to be read at line 430, this line loops back on itself until the buffer is clear.
- LINE 360: Increases the value of S by 4 in preparation for the next time through the loop.
- LINE 370: If the value of S is such that it is the first location on a new line, then S is increased by 40 to skip a line.

LINE 500: Normally, location 1744 contains the value 32, i.e. the screen code for a SPACE. Line 500 tests the contents of this location to see if an alien has reached it. Only when the contents of the location change from 32 is the program allowed to continue in order to execute the 'End of game' routine.

(3) End of game

LINE 650: Clear the screen.

LINES 630--660: Form a loop in which the screen is POKEd, first green, then red. Note the use of 'dummy' loops to slow the flashes down.

LINES 670--710: The cursor is positioned in the middle of the screen ready to print a message. Note that PRINT causes the cursor to move one line down the screen. TAB(X) moves the cursor X spaces in from the left-hand margin.

LINE 720: Clears the keyboard buffer, as previously explained.

LINE 730: Loops back on itself until the SPACE bar is pressed.

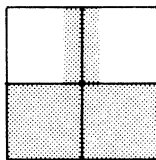
LINE 740: RUNs the program again.

(4) The subroutines

(a) *The laser base subroutine*

LINE 1010: B is the color code of the laser base.
A is the position of the laser base.
This line relates B to A.

LINES 1020 -1050: POKE the laser characters and colours to the screen.
The laser base is made up of 4 characters:



If the bottom left character is given a screen code A, then the other blocks are related as follows:

A-40	A-39
A	A+1

The POKE codes are:

108	123
160	160

Note that the code for a REVERSED character is 128+POKE code. So a reverse SPACE has a code of 128+32=160.

The color number is 2; i.e. red.

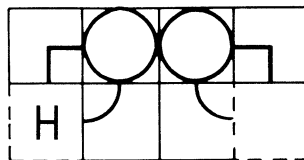
- LINES 1060- 1080: Prints two blocks of four SPACeS; one on each side of the current laser base position. This means that as the laser is moved to the right or to the left, so the old laser is rubbed out.
- LINE 1100: The GET command reads any character input at the keyboard, and this is stored in the string variable AS.
- LINE 110: IF the letter input is a 'Z', then the laser position is reduced by 2; i.e. when the laser is replotted it will appear 2 character blocks to the left.
 The second part of the line sets a lower limit on the value of A. If A were to be reduced past 1984 then the laser would appear one line up and at the right-hand side of the screen at location 1982.
- LINE 1120: Similarly if 'X' is input then the laser position is increased by 2. The upper limit for A is also set. (Beware! Weird things can start happening if you inadvertently POKE numbers into locations greater than 2023.)
- LINE 1130: If 'M' is input then the program is sent off to the subroutine, starting at line 2000.
- LINE 1140: RETURNs to the main program.

(b) Fire subroutine

- LINE 2010: Relates P to A. P is the initial position of the missile. I is set to zero.
- LINE 2020: The subroutine takes the form of a loop which starts here. Each time the loop is executed, the value of I is increased by 1.
When I reaches 22 then the missile has reached the top of the screen. If this is the case then the missile is POKEd off the screen, and the program jumps out of the loop to RETURN to the main program.
- LINE 2030: POKEs the missile color and character to position P on the screen.
- LINE 2040: Rubs out the previous missile position.
- LINE 2050: The value of P is reduced by 40; i.e. P is now given the value of a location one place up the screen. The program is sent back to line 2020 to execute the loop again.
- LINE 2060: The PEEK looks at the location directly above the current value of P. If the value in that location is not 32, i.e. a SPACE, then, firstly the missile is rubbed out, and secondly the program is sent to a subroutine at line 3000. This subroutine rubs out the alien hit.
Note: It is possible to call up a subroutine from inside another, but there is a limit to the number of tiers of subroutines that one can have 'open' at any one time.
- LINE 2090: RETURNs the program to subroutine 1000.

(c) Hit subroutine

- LINE 3010: Rubs out the missile.
- LINES 3020–3050: The location P is PEEKed. It may contain any one of 4 parts of the alien character. The value of H is set as follows:



- LINES 3070–3090: Prints a 4*2 rubout block over the alien as follows:

H-40	H-39	H-38	H-37
H	H+1	H+2	H+3

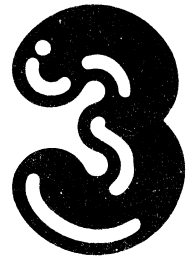
LINE 3100: RETURNS the program to subroutine 2000.

PROGRAM STRUCTURE

Note that the program is made up of three distinct parts:

- 1. A 'set up' routine.
- 2. The main program loop.
- 3. Subroutines that are called up as necessary.

Putting the laser base control onto a subroutine means that it can be called up more than once during the main program loop. This can be a useful structure as it is often the user-controlled movement that is unresponsive in games of this type written in BASIC. Sensitivity can be greatly increased by accessing user control routines at several points in the main loop.



Printing to the screen

In the last chapter we looked at making characters appear on the screen by using the POKE command. An alternative to this is to use the PRINT command. The Commodore version of PRINT is a powerful tool with many different aspects to its operation. In this chapter we shall look at these PRINT functions available to you on your Commodore 64.

PRINT AND PUNCTUATION

With some important exceptions which will be explained later, the PRINT statement in BASIC will act in two ways. Firstly

```
PRINT X
```

will cause the current value of the variable X to be printed on the screen. PRINT's second mode of operation is to copy a set of characters to the screen. These characters being placed inside quotation marks after the PRINT command. For example

```
PRINT "GOOD MORNING"
```

It is possible to combine two or more variables and/or strings of characters by the use of punctuation.

If two items to be PRINTed are separated by a SEMICOLON (;), then the second item will be positioned immediately after the first:

```
10 AS="HEL":BS="LO"
20 PRINT AS;BS
```

This program will PRINT the complete word HELLO on the screen. The two exceptions to this rule are:

- (1) Numeric items are followed by a space.
- (2) Positive numbers are preceded by a space. (In the case of negative quantities this space is used for the minus sign.)

The screen is divided vertically into zones, ten spaces wide. If a COMMA (,) is used to separate two items on a line, then the second item will be placed at the start of the next zone. This facility is especially useful forming tables of values etc. If we amend line 20 above so that the semicolon is replaced by a comma then the resulting output will look like this:

```
HEL      LO
```

If a semicolon or comma is present at the end of a list of items then this will cause the first item in any following PRINT statement to be placed on the same line and to be positioned in accordance with the punctuation. When no punctuation is present at the end then any later items will commence at the beginning of a new line.

POSITIONING OUTPUT

(1) The TAB command

The TAB command enables you to position the beginning of any output. Its format is as follows:

```
PRINT TAB(17)"CENTRE"
```

This will cause the word CENTRE to be printed starting at the eighteenth character position across the screen. The vertical position of CENTRE is determined by the prior position of the cursor. The screen is 40 character spaces wide, but the TAB number can be any integer in the range 0-255. Hence TAB(80) would commence PRINTing two lines below the old cursor position.

(2) The cursor controls

You may have already experienced some strange happenings whilst trying to edit characters inside quotation marks. When a quote (") is typed the cursor controls cease to operate in their normal way. The cursor controls are said to be in quote mode. Even though they appear as reversed characters within the quotation marks, their function is to position the cursor during a program run. For example try RUNing this program which also demonstrates the 64's internal clock, TIS.

```

5 REM***CHAP3 PROG1***
10 REM  *****
20 REM  *                *
30 REM  *  TIMER  *
40 REM  *                *
50 REM  *****
60 PRINT"␣":REM CLEAR SCREEN
70 TI$="000000"
80 PRINT TAB(16)"␣";TI$
90 GOTO 80
100 REM TO END PRESS RUN/STOP

```

Each time the program executes the loop a new print line is started. However, due to the 'cursor up' control character, the original line is over-printed with the new value of TI\$. Incidentally, the six digits of TI\$ are HHMMSS.

When using the cursor controls in quote mode it is important to bear in mind the cursor position just prior to the execution of the PRINT statement containing them. The HOME and CLR/HOME functions also operate in quote mode and can be used to help keep track of the cursor position during a program run.

Note: The program listings in this book do not use the cursor control characters. For reprographic reasons we have adopted the alternative CHR\$ format. To demonstrate this, here is the Timer program again. 'Cursor up' control is done by means of CHR\$(145). A full list of CHR\$ codes is in Appendix F of the User Manual.

```

5 REM***CHAP3 PROG2***
10 REM  *****
20 REM  *                *
30 REM  *  TIMER  *
40 REM  *  AGAIN  *
50 REM  *                *
60 REM  *****
70 PRINTCHR$(147):REM CLEAR SCREEN
80 TI$="000000"
90 PRINT TAB(16)CHR$(145);TI$
100 GOTO 90
110 REM TO END PRESS RUN/STOP

```

The full set of cursor control characters is:

Key	Appears as
	
	
	
	
	
	

(3) Color

The color of characters to be printed can also be programmed in quote mode. The color control characters may appear at any position along the PRINT line and any characters to the right will appear in that color until a new color control appears. The color controls are programmed in quote mode the number keys in conjunction with either CTRL or the special Commodore key. A full table of the colors available is on page 57 of the User Manual.

This simple program demonstrates their function.

```
5 REM***CHAP3 PROG3***
10 REM *****
20 REM *
30 REM * RAINBOW *
40 REM *
50 REM *****
60 PRINT "Q"
70 POKE53281,11:REM GREY BACKGROUND
80 PRINT "*****";REM CURSOR DOWN
90 FORI=0TO10STEP2:REM UP-
100 PRINT TAB(I)"*****"
110 NEXT
120 PRINT TAB(15)"*****"
130 FORI=20TO30STEP2:REM -AND DOWN
140 PRINT TAB(I)"*****"
150 NEXT
160 END
```

(4) The Commodore character set

On the majority of keys there are two small squares, each containing a special character. These may be PRINTed out by placing them within quotes. The right-hand character on any key is PRINTed by pressing the key and SHIFT. The left-hand one is obtained by using the key together with the Commodore flag key. For example, to create our alien friend from Chapter 2, type in this (C= is the Commodore flag key):

```
C=/A SHIFT/W SHIFT/W C=/S (RETURN)
SPACE SHIFT/K SHIFT/J
```

(5) Reverse mode

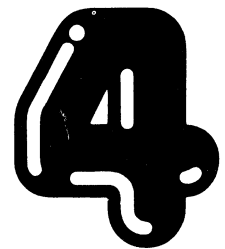
Pressing CTRL and 9 together inside quotation marks causes an inverse 'R' to appear. This is the control character for reverse video. Any characters to the right of this will appear as 'negatives' of their usual selves. Turning off reverse mode is accomplished by CTRL/0.

Using this mode it is possible to isolate any group of characters within a string and PRINT them as negatives.

This program demonstrates how reverse mode can be used to produce flashing characters.

```
5 REM***CHAP3 PROG4***
10 REM *****
20 REM *
30 REM * NEGATIVE *
40 REM *
50 REM *****
60 PRINT"R"
70 PRINT"RRRRRRRR":REM CURSOR DOWN
80 PRINT TAB(10)"I'M FEELING PRETTY"
90 PRINT TAB(15)"RRRNEGATIVE"
100 FORI=1TO500:NEXT
110 PRINT TAB(15)"RRRNEGATIVE"
120 PRINT TAB(16)"RRRTODAY"
130 PRINT"RRRR":REM CURSOR UP
140 FORI=1TO500:NEXT
150 GOTO90
160 REM TO END PRESS RUN/STOP
```

We now have two methods at our disposal for replacing and colouring characters on the screen. Each method has its advantages and disadvantages. There is no reason why both methods cannot be used within the same program to fulfil different functions. Deciding which method to employ in any given circumstances is largely dependent on experience and personal preference.



The logical operators AND, OR and NOT

THE DECIMAL SYSTEM

Before we look at the binary system, let's look at our normal system of counting – the decimal system.

In a number such as 1375 each digit has a certain value. For example, the '1' means 'one lot of a thousand'; the '3' means 'three lots of a hundred' and so on. We could arrange 1375 in columns, each column having a headline as follows:

Th	H	T	U
1	3	7	5

The largest number that may appear in any column is a 9.

The column headings are based on the number 10.

UNITS	1
TENS	10
HUNDREDS	$10 * 10 = 100$
THOUSANDS	$10 * 10 * 10 = 1000$

THE BINARY SYSTEM

Just as the decimal system is based on 10, so the binary system is based on 2. The column headings may be found as follows:

UNITS	1
TWOS	2
FOURS	$2 * 2 = 4$
EIGHTS	$2 * 2 * 2 = 8$
SIXTEENS	$2 * 2 * 2 * 2 = 16$

The largest number that may appear in any column is a 1. Hence the only digits that are used in binary are 0 and 1.

Why does a computer use binary? Well, quite simply, the number 1 can be represented by an electrical voltage. 0 is represented by an absence of such a voltage.

CONVERTING FROM BINARY TO DECIMAL

Let's convert the binary number 11001011 to decimal.

First of all we must write out the binary number complete with column headings.

128	64	32	16	8	4	2	U
1	1	0	0	1	0	1	1

All we now have to do is add up the values as follows:

$$1 * 128 = 128$$

$$1 * 64 = 64$$

$$1 * 8 = 8$$

$$1 * 2 = 2$$

$$1 * 1 = 1$$

$$\text{TOTAL} = 203$$

So 11001011 in binary is 203 in decimal.

CONVERTING FROM DECIMAL TO BINARY

This is accomplished by repeated division by 2. For example, if we take the decimal number 203, let's try to get back to its binary equivalent.

$$\begin{array}{r}
 2 \overline{) 203} \quad r1 \\
 2 \overline{) 101} \quad r1 \\
 2 \overline{) 50} \quad r0 \\
 2 \overline{) 25} \quad r1 \\
 2 \overline{) 12} \quad r0 \\
 2 \overline{) 6} \quad r0 \\
 2 \overline{) 3} \quad r1 \\
 2 \overline{) 1} \quad r1 \\
 0
 \end{array}$$

The binary equivalent of 203 is then found by reading the 'remainder' column.

REPRESENTING NUMBERS USED BY THE LOGICAL OPERATORS

Each Binary digit is known as a BIT and any decimal number is stored as a series of 16 bits. For example, the binary equivalent of 241 is 11110001. Only 8 bits are required to represent this number in binary but the computer will hold 241 as 0000000011110001, the other 8 bits being filled with zeros.

In this way numbers in the range -32768 to +32767 may be stored. If a number used with one of the logical operators is outside this range then an error message is generated.

How are negative numbers stored? Well, here life gets a little more complex. Negative numbers are stored by 'twos-complement'. Briefly, this is how it works.

For example, let's take the number -241.

We already know that +241 in 16-bit form is 00000000111100001. To represent -241 we, firstly change all the ones to noughts and vice versa, and secondly add 1 to the result.

	16-bit	Decimal
	0000000011110001	+241
change 0s and 1s	1111111100001110	
add 1	1111111100001111	-241

We now have the 'twos complement' form for -241. We must understand and appreciate these processes so that we can understand the functions offered by the logical operators.

THE OPERATOR 'AND'

You may have seen something before, in a program, which looks like this:

```
X=35AND18
```

This does not mean that $X = 35 + 19$, i.e. 54. This is a 'logical' use of the word AND. Look at the circuit shown in Fig. 4.1. The circuit consists of a battery and a bulb. Also in the circuit are two switches, S1 and S2. What arrangement of switches would be required to light the bulb? The truth table for this circuit is:

S1	S2	BULB
off	off	off
off	on	off
on	off	off
on	on	on

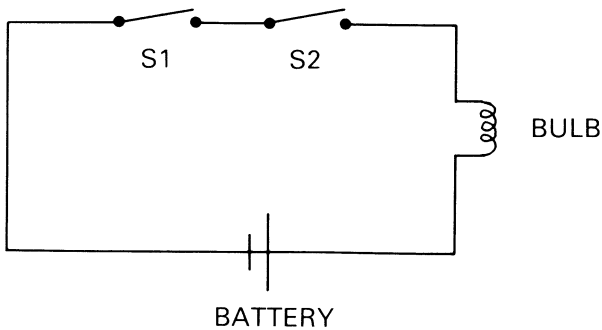


Fig. 4.1

The only way the bulb will light is if S1 AND S2 are in the 'on' position. If we use 0 to represent 'off' and 1 to represent 'on' the truth table becomes:

S1	S2	BULB
0	0	0
0	1	0
1	0	0
1	1	1

Similarly the only way to get an 'output' of 1 from the bulb is if S1 is 1 AND S2 is 1. We can use the above truth table to help us work out

$$X = 35 \text{ AND } 18$$

Firstly we have to convert 35 and 19 to their 16-bit binary representations.

	16-bit		Decimal
	0000000000100011		35
AND	0000000000010010	AND	18
	0000000000000010		2

If we take the two binary numbers column by column and only write a 1 in the answer box if both digits are 1 then we form the answer above. Converting this back to decimal gives the number 2.

So, $X = 35 \text{ AND } 18$ means that $X = 2$.

THE OPERATOR 'OR'

Look at the different electrical circuit shown in Fig. 4.2. This time the truth table will look like this:

S1	S2	BULB
0	0	0
0	1	1
1	0	1
1	1	1

Very simply, the bulb will light if either S1 OR S2 or both are closed.

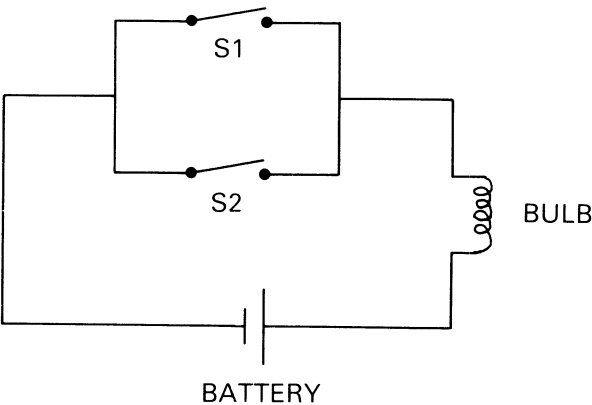


Fig. 4.2

If we now look at an example using the logical OR:

B=124OR73

16-bit		Decimal	
	0000000001111100		124
OR	0000000001001001	OR	73
	0000000001111101		125

This time we should put a 1 in the answer box if there is any combination of 1s and 0s, other than two 0s.

Converting back to decimal tells us that 124 OR 73 = 125.

THE OPERATOR ‘NOT’

Finally, let’s look at the NOT operator. Again, we can model its function by means of an electrical circuit (see Fig. 4.3). In this case the bulb will light of the switch is open and will go out if the switch is closed. The truth table, then, is very simple:

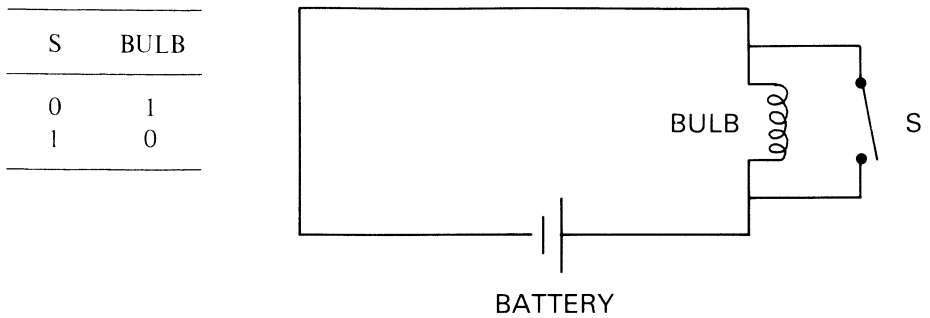


Fig. 4.3

In this way the NOT operator forms the 'twos complement' of the operand (see earlier in this chapter for an explanation of 'twos complement').

A useful expression which connects any integer X with its complement is:

$$X = -(X + 1)$$

Examples of the NOT operator:

NOT253 = -254

NOT16 = -17

NOT-89 = 88

etc.

FURTHER USES OF 'AND' AND 'OR'

The operators AND and OR can also be used, in a more familiar way, to relate two or more statements in a conditional statement. Look at the following program and before you try to run it see if you can predict the outcome.

```

10 CHR$(147):REM CLEAR SCREEN
20 FORI=1TO8
30 FORJ=1TO5
40 IFI=3ANDJ=2THEN PRINTI,J
50 NEXT: NEXT
60 END

```

The program will run through the pair of nested loops but will only print out I and J IF I is 3 and J is 2. Obviously RUNning this program will yield the result:

```
3      2
```

This seems to be a natural usage of AND. OR can be used in much the same way, either by itself or in conjunction with other operators. Amend line 40 to read

```
40 IF I=3 AND J=2 OR J=4 THEN PRINT I,J
```

RUNning the program with this new line will result in

```
1      4
2      4
3      2
3      4
4      4
5      4
6      4
7      4
8      4
```

The computer must be carrying out the AND operation first. Hence I and J will be printed out if either I = 3 and J = 2, or if J = 4.

We can change the order of priority by the use of brackets. Amend line 40 to read

```
40 IF I=3 AND (J=2 OR J=4) THEN PRINT I,J
```

This time we get the outcome

```
3      2
3      4
```

The bracket around the OR part of the statement means that this operation is carried out first.

The order of priority on the Commodore 64, when AND and OR are combined, is such that the AND is carried out first unless there are brackets present to indicate a different priority.

In this chapter we have learned several important facets of the computer's internal operation. This knowledge will be utilized at various points in the rest of the book.

5

User-defined graphics

Although some impressive results may be obtained from use of the standard Commodore character set, you will at some stage want to be able to design and use your own characters. This process is not as difficult as it sounds. In this chapter you will be taken gently through the steps needed to construct user-defined characters.

THE SUPERSHIFTER PROGRAM

Unfortunately, the standard set of characters is stored in ROM memory. This means that we cannot redesign any characters straight away. Before we can really start the design process we must copy the character set from ROM and RAM. Look at the table of screen codes in Appendix E of the User Manual. We are still going to need the alphabet set, the numbers 1 to 9, and some of the punctuation, but if we copy the Commodore characters as well we are going to start eating up the memory available for a program. All we really require is the first 64 characters (i.e. up to and including the question mark).

The SUPERSHIFTER program below is designed to do just that. SUPERSHIFTER is a BASIC LOADER for a MACHINE CODE program. We suggest that if you require user-defined characters as part of your program then you should use SUPERSHIFTER in a subroutine. Of course, being a machine code routine, SUPERSHIFTER will accomplish its task much more quickly than the correspondingly BASIC routine.

Type in SUPERSHIFTER and SAVE it for future use.

```

1000 REM  ****
1010 REM  **
1020 REM  **"SUPERSHIFTER"**
1040 REM  **  CHARACTERS  **
1050 REM  **    FROM    **
1060 REM  **  ROM TO RAM  **
1070 REM  **
1080 REM  ****
1100 POKE52,48:POKE56,48:REM -LOWER MEMTOP-
1110 FORI=828TO950:REM -TEMP STORE-
1120 READJ      :REM -----
1130 AA=AA+J      :REM -BASIC
1140 POKEI,J      :REM  LOADER-
1150 NEXT        :REM -----
1160 READJ      :REM -CHECKSUM-
1170 IFAA<>JTHENPRINT"DATA ERROR":END
1180 SYS(828)      :REM -EXECUTE-
1190 POKE53272,28 :REM -POINT TO RAM-
1200 REM -M/C DATA-
1210 DATA 169,0,141,14,220,169,51,141
1220 DATA 1,0,162,0,189,0,208,157,0,48
1230 DATA 189,0,209,157,0,49,189,0,210
1240 DATA 157,0,50,189,0,211,157,0,51
1250 DATA 189,0,212,157,0,52,189,0,213
1260 DATA 157,0,53,189,0,214,157,0,54
1270 DATA 189,0,215,157,0,55,232,224
1280 DATA 255,208,203,173,255,208,141
1290 DATA 255,48,173,255,209,141,255
1300 DATA 49,173,255,210,141,255,50
1310 DATA 173,255,211,141,255,51,173
1320 DATA 255,212,141,255,52,173,255
1330 DATA 213,141,255,58,173,255,214
1340 DATA 141,255,54,169,55,141,1,0
1350 DATA 169,1,141,14,220,169,28,141
1360 DATA 24,208,96
1370 REM -CHECKSUM-
1380 DATA 16246
1400 RETURN

```

DESIGNING YOUR OWN CHARACTERS

If you carefully study any normal character on the screen, you will see that it is made up of a pattern of small squares. These squares are called picture elements, or PIXELS. Each character is actually a square, 8 pixels wide and 8 pixels high. Some of the pixels in this 8*8 grid are turned on. These are the ones you can see. The other pixels making up the grid are turned off.

Fig. 5.1 shows how a capital 'V' is made. Notice that each row of 8 pixels can be thought of as a binary number. A bit 1 represents a pixel which is turned on and a bit 0 represents a pixel turned off. Each binary number is 8 bits, or 1 BYTE long. This can be converted to a decimal number.

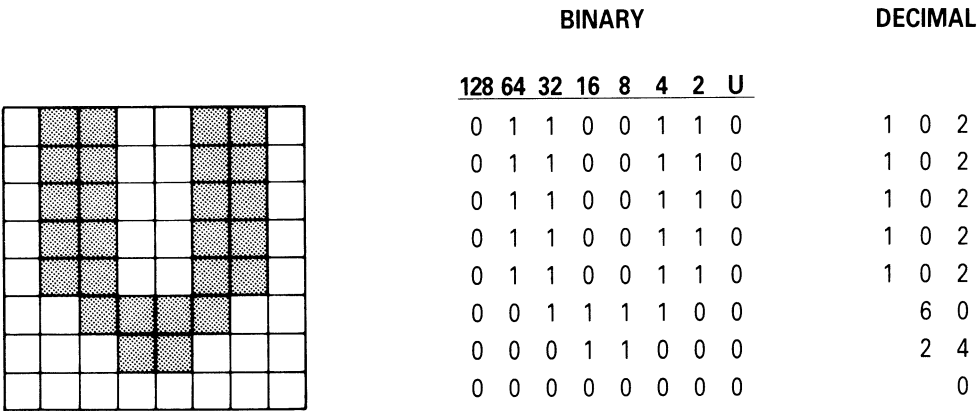


Fig. 5.1

After using SUPERSHIFTER, the character set will be held in RAM memory. The starting location for the 'shifted' character set is 12288. Each location can hold one byte of information. It takes 8 bytes to store the information required to define one character. Hence the 8 bytes defining the first character '@' will be held in locations:

- 12288
- 12289
- 12290
- 12291
- 12292
- 12293
- 12294
- 12295

The next character 'A''s data will be held in locations 12296–12303, and so on. There is a simple relationship between a character's screen code, as given in Appendix E of the User Manual and the first memory location for that character in RAM:

$$\text{1st location in RAM} = 12288 + (8 * \text{Screen Code})$$

Now let's design our own character. Firstly, we must either type in or load SUPERSHIFTER. We can now build the rest of the program around it. SUPERSHIFTER should be used as a sub-routine, starting at line 1000. After loading SUPERSHIFTER, the first line of our program will be:

```
10 GOSUB 1000:REM SUPERSHIFTER
```

We must now design our character. I have decided to draw an alien-type being on an 8*8 grid. The design is then represented in binary and the binary is converted to decimal (see Fig. 5.2). You can use my design, or make one up yourself! Eventually we end up with 8 decimal numbers which define our character. All that remains to do is to POKE these numbers into some locations in RAM. If we do this we are going to overwrite the data for one of the characters we have shifted from ROM. It is best to choose a character that is not going to be used at any stage in the program. For our purposes, the ampersand '&' seems a safe bet.

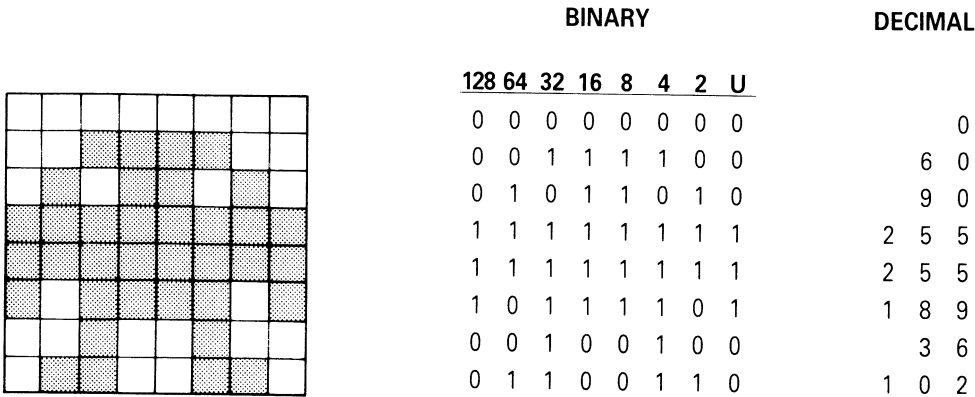


Fig. 5.2

The ampersand's screen code is 38. We can find the first location in RAM by using the relationship above.

$$\begin{aligned} \text{1st location in RAM} &= 12288 + (8 * 38) \\ &= 12592 \end{aligned}$$

We can also READ in the 8 numbers as DATA and POKE them to locations between 12592 and 12599. The following line will do this:

```
20 FOR I=12592 TO 12599:READ A:POKE I,A:NEXT
```

The data for our character should be placed at the end of the program, say at line 10000. The full program listing is:

```
1 PRINT "*****"
2 REM *
3 REM * DEFINE A *
4 REM * CHARACTER *
5 REM *
6 REM *****
7 REM
10 GOSUB 1000:REM SUPERSHIFTER
20 FOR I=12592 TO 12599:READ A:POKE I,A:NEXT
30 END
1000 REM SUPERSHIFTER
"
"
"
1999 RETURN
10000 REM CHAR DATA
10010 DATA 0,60,90,255,255,199,36,102
```

RUN the program and then try to use the ampersand (SHIFT/6). You can now use our new character in PRINT statements, and as the screen code for our alien is still 38 we can POKE it to the screen and color it in the normal way. You can, of course redefine as many of the normal 64 characters as you wish. But make sure that you don't want to use that character in its normal way.

As you can see from the example, one user-defined character on its own is rather small. However, it is possible to group a few characters together to make a larger one (see the 'Alioids' program in Chapter 2).

MULTI-COLOR MODE

Multi-color mode on the Commodore 64 allows you to use up to 4 colors inside each 8 by 8 character square. Before you can start to use multi-color however, you have to turn it on. This is

done by setting the fifth bit of location 53270 to a 1. You could just POKE in 16 to do this, but other bits in this register control other graphics functions. POKEing 16 would set all the other bits in the register to zero. A neat way to set bit to 1 without affecting the other bits is to use the following POKE. This method is more fully discussed in Chapter 6.

```
POKE 53270,PEEK(53270)OR16
```

When multi-color is switched on, the computer starts to interpret the bit patterns in a different way. Instead of considering them individually, as in normal mode, the bits are grouped horizontally into pairs. As each bit can be either on (1) or off (0), there are 4 possible combinations when the bits are paired. These are: 0 0, 0 1, 1 0, and 1 1. Each of these patterns relates to a color in multi-color mode and both bits are assigned that color. The 4 colors used can be programmed by POKEing a color code number into the appropriate location. Table 4.1 shows how the bit pairs select color and the location into which you can POKE the color code. Unfortunately, if you use multi-color mode then horizontal resolution will be reduced by a factor of 2. To illustrate the use of multi-color mode, lets look at our alien character again. Once multi-color is switched on then the bit patterns which defined the alien are interpreted as shown in Fig. 5.3. You can see that the same bit patterns in multi-color mode draw a picture that has little relation to the original alien character. Hence it is important when designing characters to bear in mind whether they are going to appear in normal or multi-color modes.

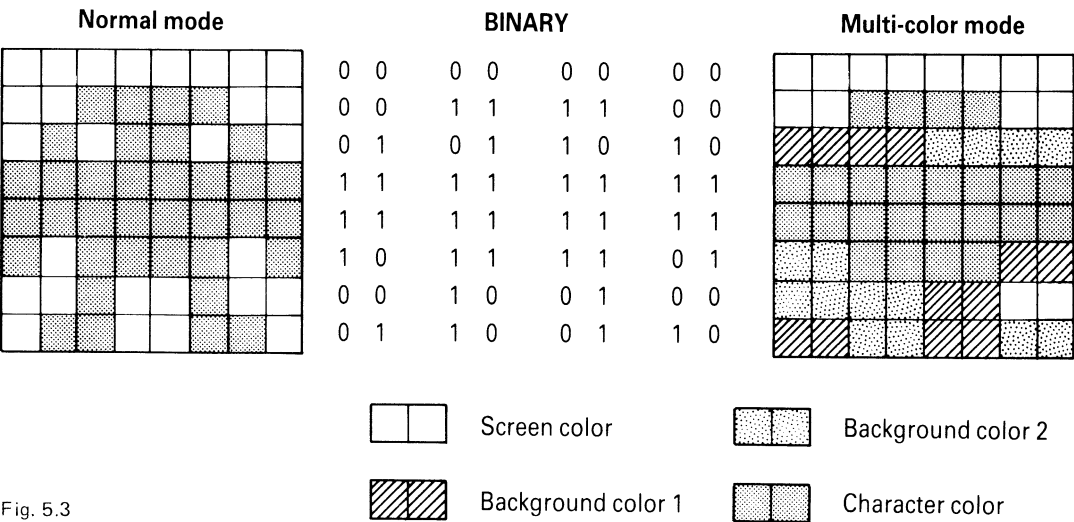


Fig. 5.3

Table 5.1

Bit pair	Color	Loocation
0 0	Normal screen color	53281
0 1	Background color 1	53282
1 0	Background color 2	53283
1 1	Normal character color	color RAM

Ordinarily, when multi-color mode is OFF, you can choose any one of 16 colours and POKE them to the screen using numbers from 0 to 15. In multi-color mode the color code is used to turn on multi-color for any one character. This is controlled by the fourth bit in each color code. If this bit is set to 1 then the character is displayed in multi-color, and if the bit is 0 then the character is displayed normally. As the color code is held in 4 bits (or nibble) this means that when multi-color mode is ON the first three bits dictate color for a character and the fourth bit dictates whether the character is displayed in normal or multi-color mode. Table 5.2 makes this clear. It can be seen from the table that if multi-color mode is turned on then color codes 0–7 cause a character to be displayed in the normal way in any one of eight colors. Color codes from 8 to 15 repeat the same eight colors but this time the character will be displayed in multi-color.

If you still have the character defining program in the computer, try making these alterations to it:

Change line 30 to:

```
30 POKE53270,PEEK(53270)OR16
```

This will turn on multi-color mode. Now add the following lines:

```
40 POKE53281,0:REM SET SCREEN TO BLACK
50 POKE53282,1:REM SET BACKGROUND COL 1 TO WHITE
60 POKE53283,5:REM SET BACKGROUND COL 2 TO GREEN
70 PRINT CHR$(147):REM CLEAR SCREEN
80 FORI=1024TO2023STEP4
90 POKEI,38:POKEI+2,38:REM POKES & TO SCREEN
100 POKE54272+I,10:POKE54272+I+2,2
110 NEXT
120 END
```

The loop from lines 90 to 110 POKEs our alien to every alternate character space on the screen. Note that when the characters are colored at line 100, the first of each pair of aliens is colored in multi-color with a red foreground. The second alien is, however, displayed in the normal way and colored red.

Table 5.2

Color code	Binary nibble	Multi-color ON		Multi-color OFF
		Display	Color	
0	0000	normal	black	black
1	0001	normal	white	white
2	0010	normal	red	red
3	0011	normal	cyan	cyan
4	0100	normal	purple	purple
5	0101	normal	green	green
6	0110	normal	blue	blue
7	0111	normal	yellow	yellow
8	1000	multi	black	orange
9	1001	multi	white	brown
10	1010	multi	red	light red
11	1011	multi	cyan	gray 1
12	1100	multi	purple	gray 2
13	1101	multi	green	light green
14	1110	multi	blue	light blue
15	1111	multi	yellow	gray 3



Sprites

Sprites are a special feature of the Commodore 64's graphics capabilities, and make action games much easier to program. They are made up of 504 pixels arranged in a rectangle 24 pixels wide by 21 pixels high. Moving characters on a screen look rather jerky. This is because they jump a block of 8 pixels each time they move from one screen location to another. Using sprites we can overcome this problem, because sprites have the ability to move one pixel at a time. Other features of sprites include priority and collision detection. The first of these means that sprites can appear to move in front of or behind other sprites. The second means that it is possible to detect either when two sprites collide or when a sprite collides with background data. More about this later. Let's get down to creating a sprite.

DESIGNING A SPRITE

The process of sprite design is very similar to that of designing user-defined characters, except that we are no longer restricted to an 8 by 8 grid. As already mentioned, a sprite is made up of a rectangular grid, 24 by 21 pixels, as shown in Fig. 6.1. Notice that the rectangle has also been divided into three columns each eight pixels wide. The first step then is to shade in the squares you want to be visible in the final sprite. Any square not shaded in will be transparent and anything in the background will show through. Fig. 6.2 shows a design of mine.

The information contained in the sprite picture has to be converted into numbers that can be read in by the computer. This is done in a similar way to the method used when dealing with user-defined characters. If we take the top row of the sprite, there are 24 pixels, divided into 3 groups of 8. As each pixel can be represented by a bit, then each group of 8 is a byte. Each byte of

[illegible]

Fig. 6.1

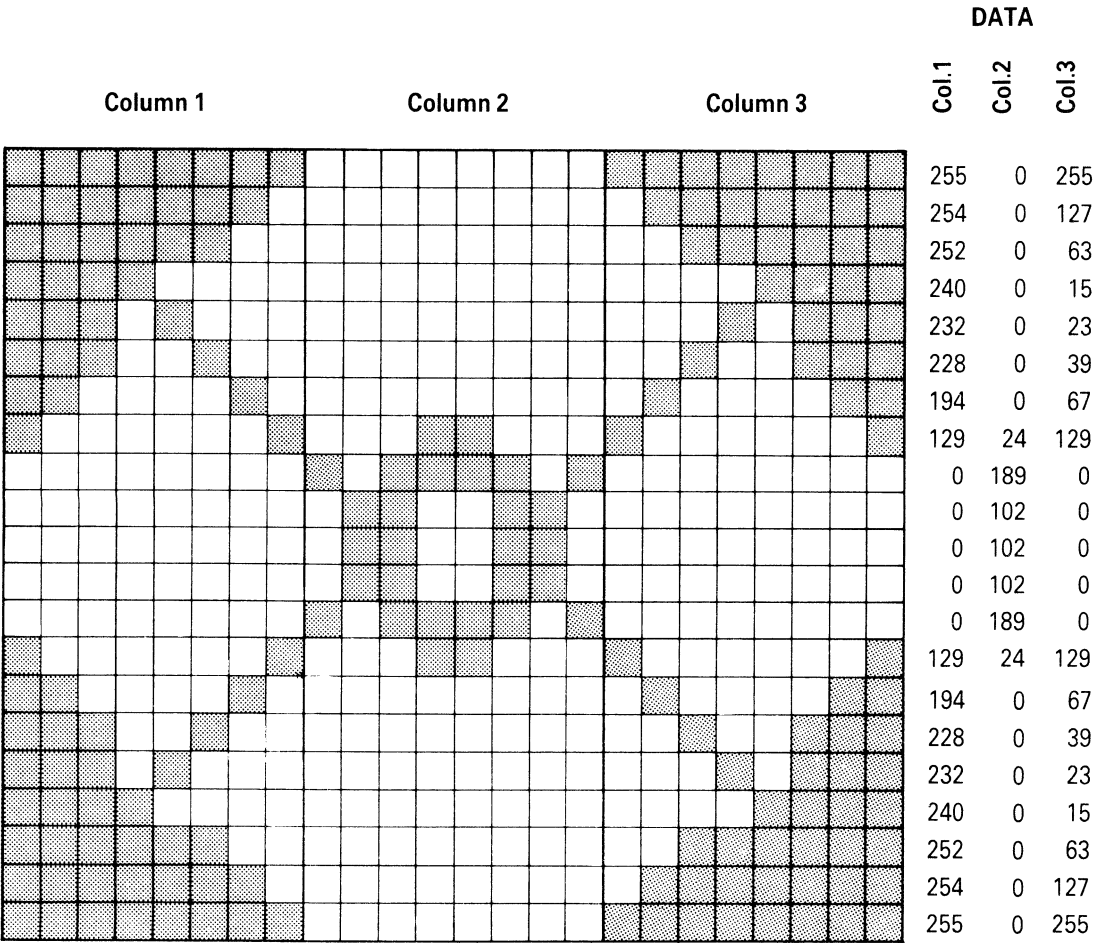


Fig. 6.2

information can be converted to a decimal number in the range 0–255. So looking at the top row again, the first byte is 11111111, which converted to decimal is 255. The second byte is 00000000, which is 0 in decimal. The third byte is again 11111111, i.e. 255. Moving to the second row, the first byte is 11111110. In decimal this is 254. If you keep working your way down through all 21 rows, you will eventually end up with 21 rows of numbers with three numbers in each row, giving you a total of 63 items of data. Now let us look at how this data can be used to form a picture on the screen.

WRITING A SIMPLE SPRITE PROGRAM

When you have designed your sprite and converted your sprite to data numbers there are still several important steps to be carried out before your sprite will appear. These are:

1. Set memory pointers.
2. Read in the data and poke it into special locations.
3. Color the sprite.
4. Position the sprite on the screen.
5. Turn on the sprite.

Each of these steps will be explained in more detail later on, so don't panic.

There are 8 sprites readily available on the Commodore 64; the sprites being numbered from 0 to 7. We are going to let our sprite be sprite 0.

The location with address 2040 holds the memory pointer for sprite 0. This pointer indicates the location at which the first piece of data for sprite 0 is held. For our purposes the memory pointer should have a value 192. To carry out step 1 then, we should POKE 2040 with 192.

We are going to store our sprite data in the 63 locations between 12288 and 12350. All we have to do to achieve this is to READ in each item of data in turn and POKE it to the corresponding location.

The color for each sprite is held in a particular location (53287 to 53294 for sprites 0 to 7). To color a sprite POKE a number between 0 and 15 into one of these locations. It is often easier to refer to locations on the VIC chip by relating them to the first location on the chip, 53248. If we call this number 'V' then the color for sprite 0 will be held in location V+39.

The position of sprite 0 is defined by its horizontal X coordinate and its vertical Y coordinate. These two coordinates define the position of the top left-hand corner of the sprite rectangle. The X coordinate is held in location V (53248), and Y coordinate is held in V+1 (53249).

Once these parameters have been set all that remains to be done is to turn on the sprite. This is done by POKEing V+21 with 1.

Complete with its lines of data and a command to clear the screen is a program that will display our sprite in a beautiful shade of black on the screen. Note that if locations are referred to as 'V+ something' then V must be defined as being equal to 53248 at the beginning of the program.

```

1 REM***CHAP6 PROG1***
2 REM  ****
3 REM  ****
4 REM  **      **
5 REM  **  SPRITE  **
6 REM  **    DEMO   **
7 REM  **    #1     **
8 REM  **          **
9 REM  ****
10 REM ****
11 PRINTCHR$(147):REM CLEAR SCREEN
20 REM SET MEM POINTER:VIS START OF VIC
30 POKE2040,192:V=53248
40 REM READ SPRITE DATA
50 FORI=12288TO12288+62:READA:POKEI,A
55 NEXT
60 REM COLOUR SPRITE BLACK
70 POKEV+39,0
80 REM POSITION SPRITE
90 POKEV,100:POKEV+1,100
100 REM TURN SPRITE ON
110 POKEV+21,1
190 REM SPRITE DATA
200 DATA255,0,255,254,0,127,252,0,63
210 DATA240,0,15,232,0,23,228,0,39
220 DATA194,0,67,129,24,129,0,189,0
230 DATA0,102,0
240 DATA0,102,0
250 DATA0,102,0
260 DATA0,189,0,129,24,129,194,0,67
270 DATA228,0,39,232,0,23,240,0,15
280 DATA252,0,63,254,0,127,255,0,255

```

Type in the program and RUN it. If the sprite doesn't look the way its supposed to then check your lines of data carefully, as it is easy to make a mistake here. Once you have got the sprite on the screen then let's play with some of the other features of sprites that we haven't touched on yet.

Try entering this command in immediate mode:

```
POKEV+29,1
```

We can expand our sprite to twice its original width using this POKE instruction. POKEing V+29 with 0 will turn this effect off. Do this and then

```
POKEV+23,1
```

This expands the sprite in the vertical direction. Of course you can expand in both directions at once. POKEV+29 with 1 again to see this.

We can recolor our sprite at will by just POKEing in another number between 0 and 15 into V+39. Try POKEV+39,7 to change the sprite's color to yellow.

Moving the sprite is also easy. POKEing V with 200 will cause the sprite to jump 100 pixels to the right. To get smooth movement across the screen enter this line in immediate mode:

```
FORX=50TO250:POKEV,X:NEXT
```

Be ready, because when you press RETURN the sprite will move smoothly but quickly across the screen. If you want a more leisurely pace then edit your line to insert a dummy loop

```
FORJ=1TO500:NEXT
```

inside the main loop. The speed of travel is determined by the length of this dummy loop. Try replacing 500 with some other numbers.

We can do the same sort of thing in the vertical direction. Try

```
FORY=50TO150:POKEV+1,Y:FORJ=1TO500:NEXT:NEXT
```

to see this. If you alter the 150 in the 'Y' loop to 250 and press return again, something curious happens. When the sprite reaches the bottom of the screen it just keeps going! This doesn't mean that we have lost the sprite, it just means that the sprite has moved out of view. It will re-emerge if you type in

```
FORY=250TO50STEP-1:POKEV+1,Y:FORJ=1TO500:NEXT:NEXT
```

This effect can be quite useful in games.

We can turn the sprite on and off without losing it by POKEing V+21 with 0. To switch it on again we just POKE V+21,1.

There are really only two ways to get rid of a sprite entirely. One is to turn the machine off, the other is to press RUN/STOP and RESTORE simultaneously.

We have now had a fairly superficial look at sprites. Now let's find out about each of the functions in more detail.

MEMORY POINTERS

As previously mentioned there are 8 sprites easily available on the Commodore 64. Each of them has a location in which to hold its memory pointer. These are as follows:

Sprite number	0	1	2	3	4	5	6	7
Location	2040	2041	2042	2043	2044	2045	2046	2047
Memory pointers	192	193	194	195	196	197	198	199

Each of the pointer numbers refer to a particular area of memory in which the data for that sprite is stored. It is possible to achieve some interesting effects by playing around with these pointers. For example, 192 points to the 63 bytes of memory starting at 12288. The pointer number 193, however, points to the 63 bytes which start at 12352. If location 2040 is POKEd with 192 then sprite 0 will get its data from the area starting at 12288. If, however, at some stage during a program it is required to change the shape of sprite 0 (say, after a collision) then this can be done simply by having the data for the shape already tucked away in locations 12352 to 12414 and switching the pointer number in 2040 to 193.

The locations which are normally used to store sprite data are:

Sprite data locations	12288 to 12350	12352 to 12414	12416 to 12478	12480 to 12542	12544 to 12606	12608 to 12670	12672 to 12734	12736 to 12798
-----------------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------

As well as these locations you can also use the cassette buffer to store some of your sprite data. The relevant locations and their memory pointers are:

Alternative memory areas

Sprite number	0	1	2
Memory pointers	13	14	15
Data locations	832 to 894	896 to 958	960 to 1022

COLORING SPRITES

Giving a sprite a single color is quite straightforward. All that is required is to POKE in the color number into the relevant location. If V = 53248 then the color register for each sprite is:

Sprite number	0	1	2	3	4	5	6	7
Color register	V+39	V+40	V+41	V+42	V+43	V+44	V+45	V+46

There is also a way of coloring your sprites in more than one color. This is done by setting the sprite to multi-color mode. There is only one multi-color register for all 8 sprites. There are 8 bits in the register and each bit corresponds to a sprite. If a particular bit is 1, then that sprite is set to multi-color mode. If the bit is 0 then the sprite is colored in the normal way.

The simplest way to set any particular bit is to POKE in a decimal number. For example, if we wished to set sprite 4 to multi-color mode then we would POKE V+28 with 16. This because 16 in binary is 1000 and so POKEing it to the multi-color register would set the fifth bit to one. This can be seen more easily with the aid of a diagram. Assuming that the register is initially set to 0:

Set sprite	7	6	5	4	3	2	1	0
	128	64	32	16	8	4	2	U
Multi-colour register	0	0	0	0	0	0	0	0
POKE with 16	0	0	0	1	0	0	0	0

Once multi-color has been set for a particular sprite then that sprite is analysed in a different way. The computer now pairs 2 pixels together and use this information to decide the color of that pair. There are 4 possible combinations of states that a pair of pixels can be in.

State		Color
0	0	Both pixels off
0	1	Right-hand pixel on
1	0	Left-hand pixel on
1	1	Both pixels on

		Transparent. Background data shows through
		Pixel pair set to MULTI-COLOUR 1
		Pixel pair set to normal SPRITE COLOUR
		Pixel pair set to MULTI-COLOUR 2

Multi-color 1 is specified by POKEing V+37 with a color code number.
Multi-color 2 is specified by POKEing V+38 with a color code number.

Unfortunately, there is only one register for each of the multi-colors to share between all the sprites. This means that you cannot have different multi-colors for different sprites displayed on the screen at the same time.

To demonstrate multi-color mode more fully here is a short program display a multi-colored sprite (see Fig. 6.3). Notice that although we have to deal with the pixels in pairs when designing our sprite, the decimal data is still calculated in the usual way. The program lisitng below uses this data to display an expanded sprite on the screen.

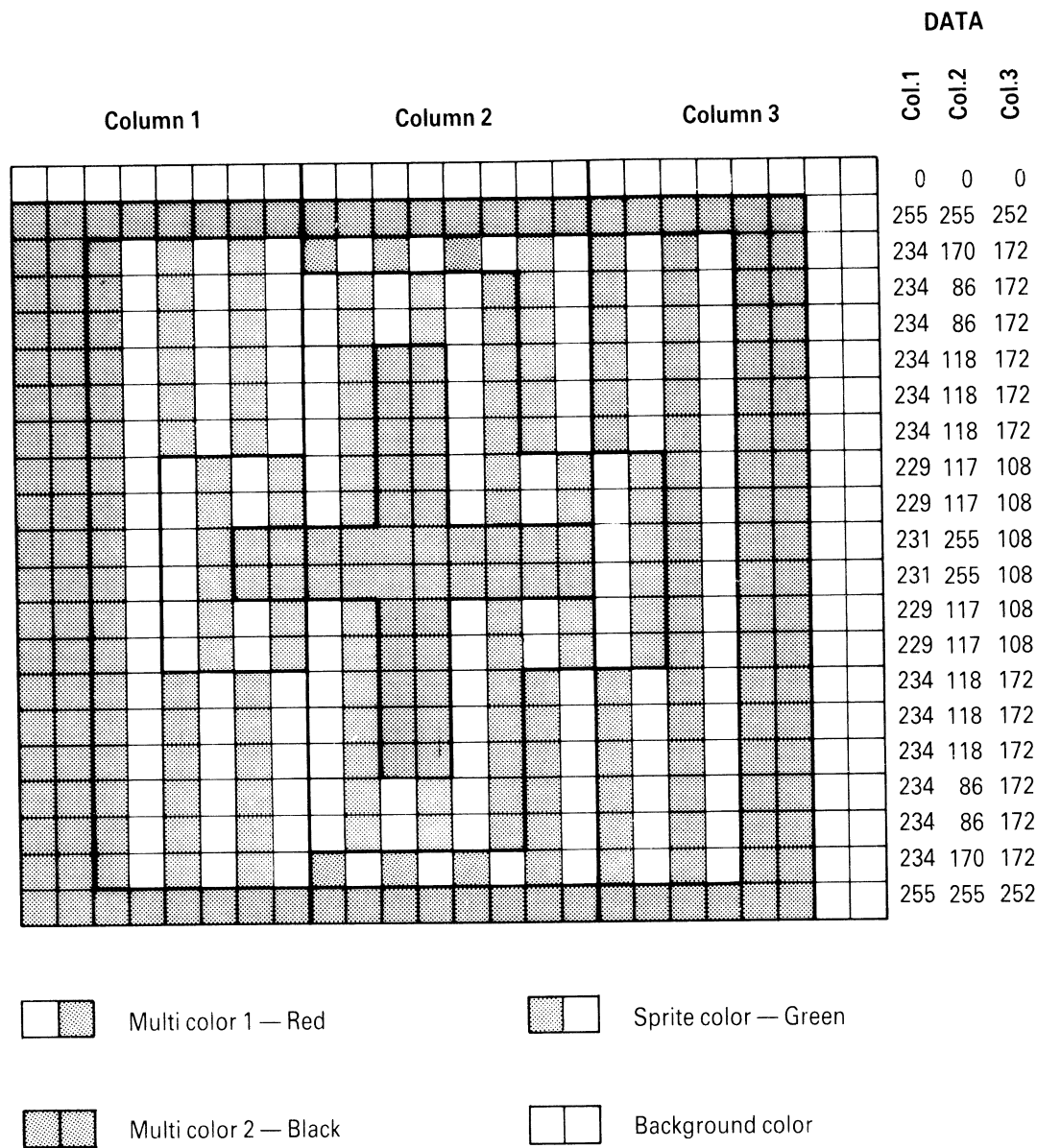


Fig. 6.3

```

0 REM***CHAP6 PROG2***
1 REM  ***
2 REM  ***
3 REM  **                               **
4 REM  ** MULTI-COLOR **
5 REM  **  SPRITE    **
6 REM  **   DEMO     **
7 REM  **                               **
8 REM  ***
9 REM  ***
10 PRINTCHR$(147):REM CLEAR SCREEN
20 V=53248
30 POKE2040,192:REM SET MEM POINTERS
40 FORI=12288TO12350:READA:POKEI,A
50 NEXT
60 POKEV+28,1:REM SET MULTI-COL MODE
70 POKEV+37,2:REM MULTI-COL 1
80 POKEV+38,0:REM MULTI-COL 2
90 POKEV+39,5:REM NORMAL SPRITE COL
100 POKEV,100:POKEV+1,100:REM SPR POSITION
110 POKEV+23,1:POKEV+29,1:REM EXPAND SPRITE
120 POKEV+21,1:REM TURN ON SPRITE
190 REM  SPRITE DATA
200 DATA0,0,0,255,255,252,234,170,172
210 DATA234,86,172,234,86,172,234,118
220 DATA172,234,118,172,234,118,172,229
230 DATA117,108,229,117,108,231,255,108
240 DATA231,255,108,229,117,108,229,117
250 DATA108,234,118,172,234,118,172,234
260 DATA118,172,234,86,172,234,86,172
270 DATA234,170,172,255,255,252

```

Whilst on the 'flag' theme the following program shows how characters POKEd to the screen can combine with spites by drawing a flagpole using reverse video SPACES (POKE code 160) and then running the sprite flag up the pole. The program below is very similar to the one used to draw the flag except that two routines have been added; one to draw the flagpole and the other to move the flag up the pole. Press RUN/STOP and RESTORE before typing in this program.

```

0 REM***CHAP6 PROG3***
1 REM  ****
2 REM  ****
3 REM  **          **
4 REM  ** FLAG AND **
5 REM  ** FLAGPOLE **
6 REM  ** DEMO     **
7 REM  **          **
8 REM  ****
9 REM  ****
10 PRINTCHR$(147):REM CLEAR SCREEN
20 V=53248
30 POKE2040,192:REM SET MEM POINTERS
40 FORI=12288TO12350:READA:POKEI,A
50 NEXT
60 POKEV+28,1:REM SET MULTI-COL MODE
70 POKEV+37,2:REM MULTI-COL 1
80 POKEV+38,0:REM MULTI-COL 2
90 POKEV+39,5:REM NORMAL SPRITE COL
100 REM DRAW FLAGPOLE
110 POKE1231,160:POKE55503,1
120 POKE1232,160:POKE55504,1
130 POKE1233,160:POKE55505,1
140 FORI=1272TO1992STEP40
150 POKEI,160:POKE54272+I,1
160 NEXT
170 POKEV+23,1:POKEV+29,1:REM EXPAND SPRITE
180 POKEV+21,1:REM TURN ON SPRITE
190 POKEV,100
200 FORY=250TO100STEP-1
210 POKEV+1,Y:NEXT
1999 REM SPRITE DATA
2000 DATA0,0,0,255,255,252,234,170,172
2010 DATA234,86,172,234,86,172,234,118
2020 DATA172,234,118,172,234,118,172,229
2030 DATA117,108,229,117,108,231,255,108
2040 DATA231,255,108,229,117,108,229,117

```

```
2050 DATA100,234,118,172,234,118,172,234
2060 DATA118,172,234,86,172,234,86,172
2070 DATA234,170,172,255,255,252
```

POSITIONING SPRITES

Sprites are positioned by means of two coordinates. The X coordinate positions the sprite horizontally and the Y coordinate sets the sprite's vertical position. Both of these numbers can have values in the range 0–255. The registers which hold the X and Y coordiantes are:

Sprite	0		1		2		3	
Coord.	X	Y	X	Y	X	Y	X	Y
Location	V	V+1	V+2	V+3	V+4	V+5	V+6	V+7

Sprite	4		5		6		7	
Coord.	X	Y	X	Y	X	Y	X	Y
Location	V+8	V+9	V+10	V+11	V+12	V+13	V+14	V+15

The normal 40 by 25 character screen in terms of pixels is 320 by 200 pixels. Y coordinates can range from 0 to 255 so there is more than enough range in the vertical direction. The problem comes when we consider the X coordinate. The largest number that can be held in any of the X registers is 255 (i.e. 11111111 in binary) but the screen runs to 320 pixels across. However, there is a way to 'open up' the right-hand side of the screen. If we had just one more bit available to us we could have numbers in the range 0 to 511, more than enough to plot across the whole screen. This extra bit is held in location V+16 and because each sprite only needs one bit to increase its X coordinate range, one of the 8 bits in location V+16 is assigned to each sprite. To make this clear, consider the example of sprite 0 moving from X = 254 to X = 257.

X coordinate	Location V+16	Location V																
254	<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>	0	0	0	0	0	0	0	0	<table><tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td></tr></table>	1	1	1	1	1	1	1	0
0	0	0	0	0	0	0	0											
1	1	1	1	1	1	1	0											
255	<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>	0	0	0	0	0	0	0	0	<table><tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr></table>	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0											
1	1	1	1	1	1	1	1											
256	<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td></tr></table>	0	0	0	0	0	0	0	1	<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1											
0	0	0	0	0	0	0	0											
257	<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td></tr></table>	0	0	0	0	0	0	0	1	<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td></tr></table>	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1											
0	0	0	0	0	0	0	1											

As the X coordinate is required to increase past 255, at the 256th position the extra bit for sprite 0 is set to 1 and the normal X coordinate register is set back to zero. This is similar to the change that occurs when 1 is added to 999 to give is 1000. The first three digits have been reset to zero and the next digit is set to 1.

In simple terms then, to plot past the 255th position POKE the V+16 register with the appropriate number and start the normal X coordinate from zero again. The correction numbers to POKE in to the V+16 register are:

Sprite	0	1	2	3	4	5	6	7
Poke	1	2	4	8	16	32	64	128

It is possible to move out of the screen area as demonstrated earlier. The origin for the X and Y coordinates does not, in fact, start at the top left-hand corner of the screen but at a position off the screen. To help you to position your sprites Figs. 6.4 and 6.5 illustrates some of the key positions, firstly for ordinary sprites and then secondly for sprites expanded in both directions.

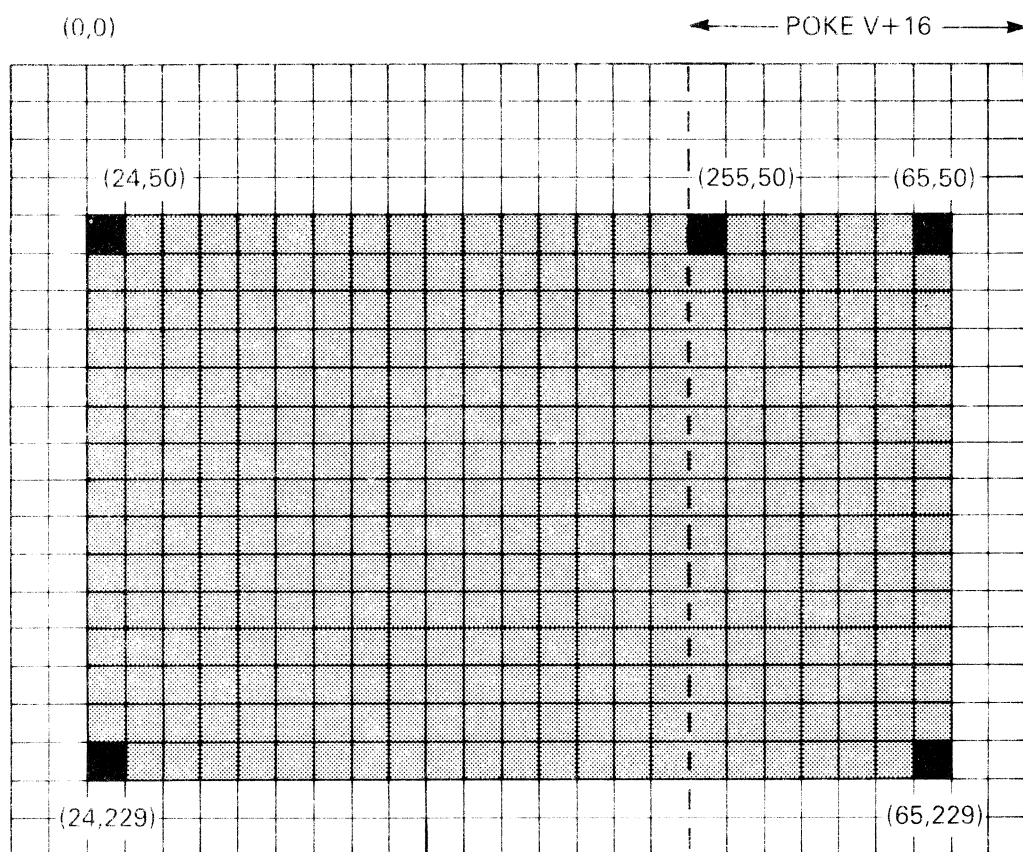


Fig. 6.4

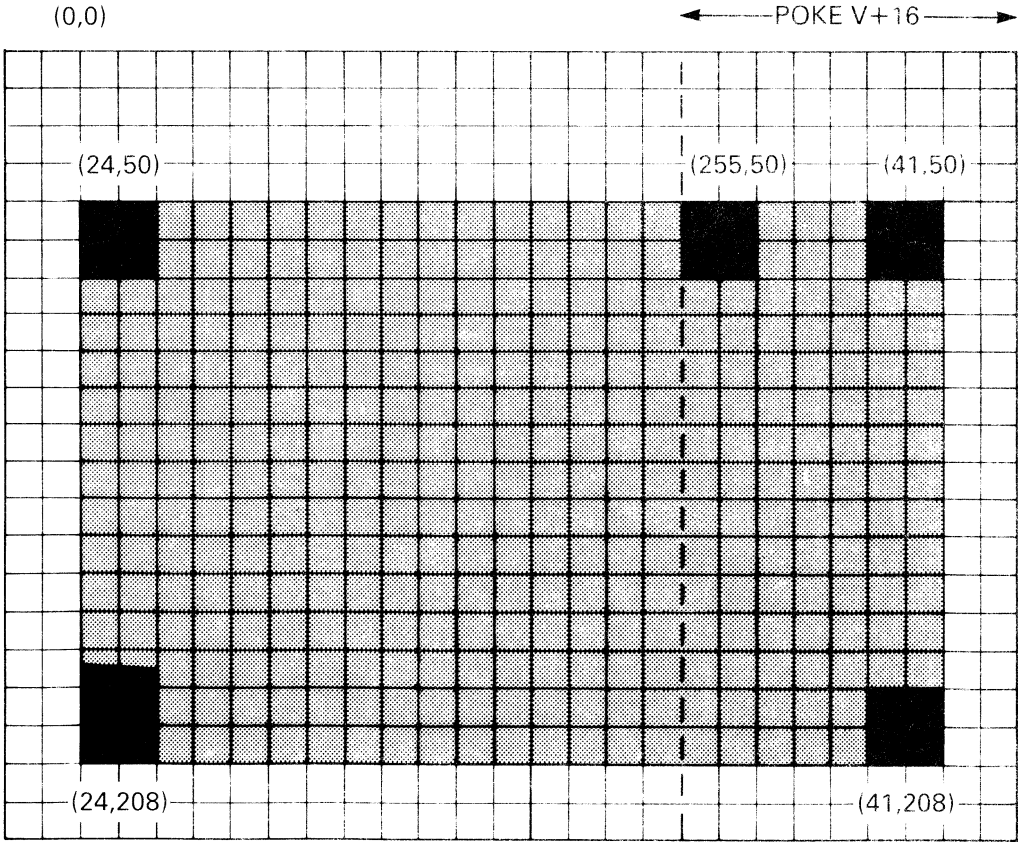


Fig. 6.5

OTHER SPRITE FUNCTIONS

All other sprite functions are controlled by having a single register for each function and one bit inside each register for each sprite. The bit furthest to the right in the register switches the function on or off for sprite 0. The bit on its left switches the function on or off for sprite 1, and so on, up to the eighth bit which controls the function for sprite 7. Functions not yet mentioned which behave in this way are:

Function	Register
Turn sprites ON/OFF	V+21
Expand in the X direction	V+29
Expand in the Y direction	V+23

USING 'AND' AND 'OR' TO ISOLATE BITS

Often programming using these single register functions can cause problems when the function is being used simultaneously with more than one sprite. The most obvious example is using the V+21 register to turn sprites on and off. Lets assume that we already have sprite 0 turned on and we now want to turn on sprite 2. For sprite 0 to be turned on the V+21 register must look like this:

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

To turn on sprite 2 we have learned that we must POKE in 4 (0000100). Unfortunately if we POKE in 4 we will not only turn on sprite 2 but we shall also turn OFF sprite 0. To get both sprite 0 and sprite 2 turned on at the same time we need to POKE in the decimal equivalent of

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

This is 5 in decimal. 1 and 4 are the numbers that turn on sprite 0 and sprite 2. To turn them both on together all we have to do is add the two numbers together. Similarly, to turn on sprite 1 requires the number 2, so to turn on sprites 0, 1, and 2 together we would need to POKE V+21 with 7 (i.e. 1+2+4).

This is fairly straightforward if you can predict which sprites are already on when you wish to turn on another, but unfortunately, in games type programs this is not always possible. However, we can still get round the problem by using the logical operators AND and OR (see Chapter 4).

Let's say that at some stage in a program we want to turn on sprite 4 without affecting any other sprites which may already be on. We can do this by PEEK'ing the $V+21$ register and performing the logical OR with the number that turns on sprite 4. This is what happens inside the register:

This is the state of the register:

Sprites 0, 1, 5, and 6 are turned on:

0	1	1	0	0	0	1	1
---	---	---	---	---	---	---	---

If we PEEK the register and OR it with 16, we get:

OR	0	1	1	0	0	0	1	1
	0	0	0	1	0	0	0	0
	0	1	1	1	0	0	1	1

`POKE V+21,PEEK(V+21)OR16`

Then the $V+21$ register will contain:

0	1	1	1	0	0	1	1
---	---	---	---	---	---	---	---

Sprites 0, 1, 5, and 6 are still turned on but so is sprite 4.

To turn off sprite 4, we must use the AND operator in such a way that only the fifth bit in the register is set to zero.

If we PEEK the register and AND it with 239 we get:

AND	0	1	1	1	0	0	1	1
	1	1	1	0	1	1	1	1
	0	1	1	0	0	0	1	1

`POKE V+21,PEEK(V+21)AND239`
returns the register to its original state:

0	1	1	0	0	0	1	1
---	---	---	---	---	---	---	---

The number 239 was found by calculating $255 - 16$.

To summarize:

If 'S' is the sprite number, then to turn ON sprite S:

$\text{POKEV}+21, \text{PEEK}(\text{V}+21) \text{OR} (\text{S} \div 2)$

To turn OFF sprite S:

$\text{POKEV}+21, \text{PEEK}(\text{V}+21) \text{AND} (255 - \text{S} \div 2)$

This method can of course be used with any of the single register sprite functions.

PRIORITY

(1) Sprite/sprite priority

When two sprites are at the same position on the screen then one appears to be on top or in front of the other. The sprite in front is said to have a higher priority than the one behind. This order of priority is not selectable by the programmer but depends on sprite number. Very simply, the rule is that lower-numbered sprites have priority over higher-numbered sprites. So if the paths of sprite 1 and sprite 2 crossed each other, then sprite 1 would appear to pass in front of sprite 2. If sprite 3's path was also crossed at the same instant then it would lie behind sprites 1 and 2. If you write programs that involve sprites passing over each other, then think carefully about which sprite you wish to appear in front and assign it a low sprite number. Sprite priority can also be used to design figures that involve multiple sprites. If there are any 'holes' in the sprite which appears in front, then the sprite behind will show through. Some interesting effects of shape and color can be achieved by layering sprites.

(2) Sprite/background data priority

Sprite to background data priority IS programmable by using the V+27 register. The function works in the same way as the other single register functions in that a bit is assigned to control the function for each sprite. If a particular bit is 0 then the sprite to which it relates will appear in front of the background. If the bit is 1 then the sprite appears to be behind the data. On power up this register is set to zero so unless you do something to change the situation all sprites will appear in front of the background.

COLLISIONS, COLLISIONS

(1) Sprite/sprite collisions

Sprite to sprite collisions are stored, when detected, in the register V+30. Each sprite has a bit in this register which is set to 1 if that sprite has been in collision with another sprite. Obviously, if two sprites are in collision then both the corresponding bits will be set to 1. To use this in your program you have to PEEK the location to see what it contains. This is normally done from inside an IF... THEN statement, as follows:

```
IF PEEK(V+30)AND$12=$12 THEN GOSUB XXX
```

where S is the sprite number. The AND isolates the particular bit in the register that relates to sprite S. You may want, as in the example, to have some action taking place because of the collision. This may be placed in a subroutine starting at XXX. If you wish to specify detection of a collision between two particular sprites S1 and S2, this can be done using this command:

```
IF PEEK(V+30)AND($12+$212)=$12+$212 THEN GOSUB XXX
```

Note: Once the collision register is PEEKed it is automatically reset to zero. If you want to use the value that you PEEKed from it in another part of the program then store the value in a variable until you wish to use it.

We found that when using this register that some strange things happened. We eventually came to the conclusion that these weird happenings were due to the register not clearing itself quickly enough after it had been PEEKed. The problems certainly cleared up when we inserted a POKE into the main action loop to 'manually' reset the register.

(2) Sprite/background collisions

The sprite to background collision register is at V+31. It works in similar way to the sprite/sprite collision register. Each sprite has a bit in the register which is set to 1 if the sprite has been in collision with background data. The register also clears itself once it has been PEEKed.

It is possible to select which piece of the background you wish to collide with. If the background is set up in multi-color mode then, although characters colored in multi-color 1 appear on the screen, they are not 'solid' sprites. On other words, if a sprite collides with a piece of data that is in multi-color 1, then that collision is not noted in the collision register. (See Chapter 2 for an explanation of background multi-color mode.)

'Aeroplane' program

The following program demonstrates the use of priority, memory pointers, and collision registers:

```

1 REM      .....
2 REM      .....
3 REM      .....
4 REM      ..... AEROPLANE .....
5 REM      .....
6 REM      .....
7 REM      .....
10 PRINTCHR$(147)
100 REM-----READ SPRITE DATA-----
110 REM READ SPRITE 0A DATA INTO 12288--
120 FORI=12288TO12350:READA:POKEI,A:NEXT I
130 REM READ SPRITE 0B DATA INTO 832--
140 FORI=832TO894:READA:POKEI,A:NEXT I
150 REM READ SPRITE 0C DATA INTO 896--
160 FORI=896TO958:READA:POKEI,A:NEXT I
170 REM READ SPRITE 0D DATA INTO 960--
180 FORI=960TO1022:READA:POKEI,A:NEXT I
190 REM READ SPRITE 0E DATA INTO 12352--
200 FORI=12352TO12414:READA:POKEI,A:NEXT I
210 Y=53248:GOSUB1000:REM SET UP SCREEN
220 REM-----COLOUR SPRITE-----
230 POKEV+39,0
240 REM-----SET Y COORD-----
250 POKEV+1,120
260 REM-----TURN ON SPRITE 0-----
270 POKEV+21,1
280 REM-----EXPAND SPRITE-----
290 POKEV+23,1:POKEV+29,1
300 REM-----MAIN JOURNEY LOOP-----
310 REM--SET POINTER TO LOOK AT 0A
320 POKE2040,192
330 FORX=70TO250:POKEV,X
340 IFL=SANDPEEK(V+31)AND1=1THEN500
350 NEXT X
360 REM--SET POINTER TO LOOK AT 0D
370 POKE2040,15:FORJ=1TO500:NEXT J

```

Listing continued next page

```

380 REM--SET POINTER TO LOOK AT 0B
390 POKE2040,12
400 REM--SET BACKGROUND TO PRIORITY---
410 POKEV+27,1
420 FORX=250TO700STEP-1:POKEV,X
430 FORJ=1TO10:NEXT:NEXT
440 REM--SET POINTER TO LOOK AT 0C---
450 POKE2040,14:FORJ=1TO500:NEXT
460 REM--RESET SPRITE PRIORITY---
470 POKEV+27,0
480 REM--COUNT THE LOOPS--
490 L=L+1
500 REM--ENSURE COLLISION REG CLEARED---
510 POKEV+31,0
520 REM--RESTART LOOP--
530 GOTO300
600 REM-----CRASH ROUTINE-----
610 REM--RESET POINTER TO LOOK AT 0E--
620 POKE2040,193
630 REM--SLIDE DOWN WALL---
640 FORY=120TO180:POKEV+1,Y
650 FORJ=1TO10:NEXT:NEXT
660 END
1000 REM-----SET UP SCREEN-----
1010 FORI=1083TO1163STEP40:POKEI,103:POKEI+1,101
1020 POKE54272+I,7:POKE54272+I+1,7
1030 NEXT
1035 REM--GRASS--
1040 FORI=1744TO1983:POKEI,160
1050 POKE54272+I,5:NEXT
1055 REM--ROOF---
1060 POKE1203,160:POKE1204,160
1070 FORJ=0TO3:POKE1242+J,160:NEXT
1080 POKE1291,233
1090 FORJ=0TO3:POKE1282+J,160:NEXT
1100 POKE1286,223
1110 FORI=1201TO1281STEP40:FORJ=0TO5

```

```

1120 POKE54272+I+J,7:NEXT:NEXT
1125 REM---WALLS---
1130 FORI=1321TO1881STEP40:FORJ=0TO5
1140 POKEI+J,160:POKE54272+I+J,7:NEXT:NEXT
1145 REM---WINDOWS---
1150 FORI=1321TO1801STEP80
1160 POKEI+1,32:POKEI+4,32:NEXT
1165 REM---ROADS---
1170 FORI=1984TO2023:POKEI,160
1180 POKE54272+I,11:NEXT
1190 FORI=1923TO1963STEP40
1200 POKEI,160:POKEI+1,160
1210 POKE54272+I,11:POKE54272+I+1,11
1220 NEXT
1230 FORI=1797TO1807STEP10
1240 FORJ=0TO3:POKEI+J,160
1250 POKE54272+I+J,11:NEXT:NEXT
1260 FORJ=1837TO1849STEP12
1270 FORI=JTOJ+120STEP40
1280 POKEI,160:POKEI+1,160
1290 POKE54272+I,11:POKE54272+I+1,11
1300 NEXT:NEXT
1999 RETURN
10000 REM----DATA FOR SPRITE 0A----
10010 DATA0,0,0,0,0,0,0,0,0
10020 DATA30,0,0,15,0,0,7,128,0
10030 DATA3,192,0,193,224,1,224,240
10040 DATA1,224,236,1,255,233,252
10050 DATA127,255,255,7,255,86,0,255
10060 DATA253,0,15,129,0,7,193,0
10070 DATA3,224,0,1,240,0,0,248
10080 DATA0,0,124,0,0,62
10100 REM----DATA FOR SPRITE 0B----
10110 DATA0,0,0,0,0,0,0,0,0
10120 DATA240,0,0,120,0,0,60,0,0
10130 DATA158,0,3,143,0,7,135,176

```

Listing continued next page

```

10140 DATA7,63,151,255,255,255,254
10150 DATA106,255,224,191,255,0,128
10160 DATA248,0,128,124,0,0,62,0
10170 DATA0,31,0,0,15,128,0,7,192
10180 DATA0,3,224,0,0,0
10200 REM-----DATA FOR SPRITE 0C-----
10210 DATA0,0,0,0,0,0,0,0,0,0,0,0,0
10220 DATA8,0,0,8,0,0
10230 DATA0,0,0,42,0,1,200,224,0
10240 DATA136,128,127,34,127,1,8,64
10250 DATA1,34,64,0,73,0,0,128
10260 DATA128,0,34,0,0,8
10270 DATA0,0,0,0,0,0,0,0,0,0,0,0,0
10300 REM-----DATA FOR SPRITE 0D-----
10310 DATA0,0,0,0,0,0,0,0,0,0
10320 DATA0,0,0,0,0,0,0,0,0,0
10330 DATA16,0,0,84,0,0,16,0,1
10340 DATA57,0,1,254,0,255,255,255
10350 DATA0,56,0,0,130,0,0,40,0
10360 DATA0,0,0,0,0,0,0,0,0,0
10370 DATA0,0,0,0,0,0,0,0,0,0
10400 REM-----DATA FOR SPRITE 0E-----
10410 DATA0,0,0,8,0,0,28,0,0
10420 DATA56,0,0,120,0,0,240,240
10430 DATA0,126,120,2,15,176,1,3
10440 DATA230,137,12,245,220,30,127
10450 DATA255,0,63,214,0,31,253,0
10460 DATA7,201,0,3,226,0,1,240,0
10470 DATA7,224,0,31,0,0,124
10480 DATA0,0,0,0,0,0,0,0

```


TABLE OF 'VIC' CHIP LOCATION FUNCTIONS

Location	Location rel. to 53248 (V)	Function
53248	V	X coordinate of sprite 0
53249	V+1	Y coordinate of sprite 0
53250	V+2	X coordinate of sprite 1
53251	V+3	Y coordinate of sprite 1
53252	V+4	X coordinate of sprite 2
53253	V+5	Y coordinate of sprite 2
53254	V+6	X coordinate of sprite 3
53255	V+7	Y coordinate of sprite 3
53256	V+8	X coordinate of sprite 4
53257	V+9	Y coordinate of sprite 4
53258	V+10	X coordinate of sprite 5
53259	V+11	Y coordinate of sprite 5
53260	V+12	X coordinate of sprite 6
53261	V+13	Y coordinate of sprite 6
53262	V+14	X coordinate of sprite 7
53263	V+15	Y coordinate of sprite 7
53264	V+16	Most Significant Bit register for sprites 0- 7 X coordinate
53269	V+21	Turns sprites 0-7 ON/OFF
53271	V+23	Expands sprites 0-7 in Y direction
53275	V+27	Selects sprite/data priority
53276	V+28	Switches on multi-color mode
53277	V+29	Expands sprites 0- 7 in X direction
53278	V+30	Sprite to sprite collision register

Location	Location rel. to 53248 (V)	Function
53279	V+31	Sprite to data collision register
53285	V+37	Sprite multi-color 1
53286	V+38	Sprite multi-color 2
53287	V+39	Color of sprite 0
53288	V+40	Color of sprite 1
53289	V+41	Color of sprite 2
53290	V+42	Color of sprite 3
53291	V+43	Color of sprite 4
53292	V+44	Color of sprite 5
53293	V+45	Color of sprite 6
53294	V+46	Color of sprite 7



Using the joystick

INTRODUCTION

On the right-hand side of your Commodore 64, next to the ON/OFF switch are two ports for joysticks. These ports are numbered 1 and 2. Plug your joystick into port 2 and type in the following program.

```
10 PRINT CHR$(145);PEEK(56320)
20 GOTO 10
```

RUN the program and you will see a number displayed on the screen. Now use your joystick. Move it up and down, and left and right. Try pressing the fire button. Notice how the number changes as you move the joystick. (If the number stays the same check that you are plugged into the correct port!)

So what does this little program do? Well, line 10 displays the contents of location 56320. There are four switches to control direction and another one for the fire button. Each of these switches controls a bit within location 56320. With the program still running, verify that the numbers opposite correspond to movements on the joystick.

Games port 2

Joystick	Decimal	Binary
None	127	01111111
Up	126	01111110
Down	125	01111101
Left	123	01111011
Right	119	01110111
Fire	111	01101111

Notice that when any switch is 'made' then the corresponding bit in the register is set to zero. Diagonal movements are formed by combining two of the four direction switches. This gives four further modes:

Up & Left	122	01111010
Up & Right	118	01110110
Down & Left	121	01111001
Down & Right	117	01110101

Break into your program and use the cursor to edit line 10 so that the location being PEEKed is altered to 56321. Plug your joystick into port 1 and RUN the program. The number displayed initially is 255 (11111111 in binary). Again this number will change as you play with the joystick. Verify that these numbers correspond to the correct direction.

Games port 1

Joystick	Decimal	Binary
None	255	11111111
Up	254	11111110
Down	253	11111101
Left	251	11111011
Right	247	11110111
Fire	239	11101111
Up & Left	250	11111010
Up & Right	246	11110110
Down & Left	249	11111001
Down & Right	245	11110101

Break into your program again. When you receive the prompt 'READY', twiddle the joystick again. You should see some rubbish appearing on the screen. This only happens when port 1 is used so it is advisable, if you write a game that needs only one joystick, to use port 2, rather than port 1.

Let's recap then on what we have found out about how the joystick works.

The joystick has five switches, four of which control movement, and one of which controls firing. There are two ports in the Commodore 64 to receive joysticks. Each port has a location

within the machine. Port 2 is at 56320 and port 1 is at 56321. The four direction switches on the joystick are linked to the first four bits in the location. The fifth bit is linked to the button. If a switch contact is made then the corresponding bit in the register is set to 0. As soon as the contact is broken then the bit's value is restored to 1. Diagonal movement is signalled by two direction switches being made simultaneously.

A BASIC JOYSTICK SUBROUTINE

It is rather inconvenient to have to refer to large decimal numbers when we want a joystick to do something inside a program. As all the directions are signalled by the first four bits of the port location, let's isolate them using AND. Firstly, we need to store the value of the location in a variable which shall be called PJ, for 'peek joystick'. If our subroutine starts at line 2000:

```
2000 REM JOYSTICK SUBROUTINE
2010 PJ=PEEK(56320)
2020 DJ=15-(PJAND15)
2030 FB=PJAND16
2040 RETURN
```

Performing the operation PJAND15 isolates the first four bits of PJ. Subtracting this from 15 gives us numbers for the direction in the range 0–10.

Line 2030 isolates the fifth bit of PJ. This bit relates to the fire button (FB). If the fire button is not pressed, then the value of this bit is 1 and FB = 16. If the button is pressed, then FB = 0.

Values for DJ relate to these directions:

DJ	Direction
0	--
1	Up
2	Down
3	--
4	Left
5	Up & Left
6	Down & Left
7	--
8	Right
9	Up & Right
10	Down & Right

The subroutine might be used inside a program like this:

```

1 REM      * * * * *
2 REM      * * * * *
3 REM      * * * * *
4 REM      * * * * * BASIC * * * * *
5 REM      * * * * * JOYSTICK * * * * *
6 REM      * * * * * DEMO * * * * *
7 REM      * * * * *
8 REM      * * * * *
9 REM      * * * * *
10 PRINTCHR$(147)
100 REM-----CREATE SPRITE-----
110 V=53248:X=150:Y=140:REM VARIABLES
120 POKE2040,192:REM SET MEM POINTER
130 FOR I=12288TO12350:POKEI,255:NEXT
140 POKEV+39,7:REM COLOUR SPRITE YELLOW
150 POKEV+21,1:REM TURN ON SPRITE
200 REM-----MAIN MOVEMENT LOOP-----
210 GOSUB2000:REM PEEK JOYSTICK
220 IFDJ=1THENY=Y-1:IFY<50THENY=50
230 IFDJ=2THENY=Y+1:IFY>229THENY=229
240 IFDJ=4THENX=X-1:IFX<24THENX=24
250 IFDJ=8THENX=X+1:IFX>320THENX=320
260 HX=INT(X/256):LX=X-256*HX
270 POKEV+16,HX
280 POKEV,LX:POKEV+1,Y:REM PLOT NEW COORDS
290 IFFB=16THEN200:REM REPEAT LOOP
300 PRINTCHR$(147);TAB(13)"BOMBS AWAY!!"
310 FORI=1TO1000:NEXT:REM DELAY
320 PRINTCHR$(147)
330 GOTO200:REM REPEAT LOOP
999 END

2000 REM JOYSTICK SUBROUTINE
2010 PJ=PEEK(56320)
2020 DJ=15-(PJAND15)
2030 FB=PJAND16
2040 RETURN

```

The main action loop starting at line 200 consists of a series of questions and action taken accordingly. For example, line 220 asks if the value of DJ is 1. This is the value of DJ that corresponds to 'Up' on the joystick. If the answer is 'yes' then the current value of the Y coordinate is reduced by 1. When the new X and Y are plotted at line 260 this will mean that the sprite has been moved 1 pixel up the screen. Note that there is a further conditional statement which sets the minimum value of Y. It is a good idea to put the boundary conditions here because if the first part of the line is not true then the computer will not test the second, but skip straight to the start of the line below. This can save on execution time as conditions are not tested needlessly. The boundaries that have been set are the edge of the screen for an unexpected sprite. As X and Y can be anywhere in the range 0–255 you can move your sprite out of the viewing area. Try setting the lower limits of X and Y to zero and the upper limit of Y to 255. RUN the program again and you will now be able to use the joystick to manoeuvre the sprite on and off the screen. You can even have your sprite going off one side of the screen and re-emerging on the other. Change the second part of line 220 to:

```
IFY<29 THEN Y=250
```

and change the second part of line 230 to:

```
IFY>250 THEN Y=29
```

The program moves the sprite, one pixel at a time, in any direction. This means that the motion is very smooth but a little slow. You can speed things up by making the sprite move more than one pixel by changing $Y=Y-1$ to $Y=Y-2$ etc. in lines 220–250. There is, however, a direct trade-off between the speed at which the sprite will move and the smoothness with which it will move. Try experimenting with the number of pixels that the sprite moves.

The program is structured in such a way at line 270 that if the value of FB is 16 then the loop is restarted (remember that $FB = 16$ if the fire button is not pressed). The computer is only allowed through to execute the lines following 270 if the fire button has been pressed, i.e. $FB < > 16$. All the program does is to print up a small message on the screen, which is held for a short time before being erased. Note that whilst the message is on the screen, the sprite cannot be moved, because the computer is tied up executing the dummy loop at line 290. The moral is, don't let your 'thing' routines be too long or you will hold up the rest of the action. Well so much for a BASIC subroutine. You can now write programs where action is controlled using a joystick, but pretty soon you are going to run into time problems, as your programs become more complex. We need to use machine code to speed things up.

A MACHINE-CODE JOYSTICK SUBROUTINE

This is a BASIC loader for the machine-code joystick routine. Because it is in BASIC, you can type it straight into your Commodore 64. To ensure that the DATA statements are entered correctly there is a checksum at the end of the subroutine. As the DATA is read in, a running total is kept. The final part of the subroutine checks the final total. If you get the message 'checksum error' then you have not entered the DATA correctly.

The joystick subroutine is numbered from 2000 onwards. Once you have entered the routine correctly, it is good idea to SAVE it to disk or tape. When you want to write a program which uses it then LOAD the subroutine first and construct the rest of the program around it.

```

2000 REM *****
2010 REM  **                               **
2020 REM  ** BASIC LOADER **
2030 REM  **   FOR M/C   **
2040 REM  ** JOYSTICK   **
2050 REM  **                               **
2060 REM  *****
2100 DATA120,173,20,3,141,29,192,173,21
2110 DATA3,141,30,192,169,71,141,20,3
2120 DATA169,192,141,21,3,88,96,120,173
2130 DATA29,192,141,20,3,173,30,192,141
2140 DATA21,3,88,96,173,27,192,205,16
2150 DATA192,240,3,76,65,193,169,255
2160 DATA141,27,192,162,0,160,0,185,2
2170 DATA220,141,28,192,169,224,153,2
2180 DATA220,185,0,220,72,173,28,192
2190 DATA153,2,220,189,8,192,221,21,192
2200 DATA189,9,192,253,22,192,144,18
2210 DATA189,8,192,157,21,192,189,9,192
2220 DATA157,22,192,104,74,72,76,162
2230 DATA192,104,74,72,176,13,222,21
2240 DATA192,189,21,192,201,255,208,3
2250 DATA222,22,192,189,21,192,221,12
2260 DATA192,189,22,192,253,13,192,144
2270 DATA18,189,12,192,157,21,192,189
2280 DATA13,192,157,22,192,104,74,72,76

```



```

2290 DATA207,192,104,74,72,176,3,254,21
2300 DATA192,208,3,254,22,192,189,0,192
2310 DATA221,17,192,189,1,192,253,18
2320 DATA192,144,18,189,0,192,157,17
2330 DATA192,189,1,192,157,18,192,104
2340 DATA74,72,76,1,193,104,74,72,176
2350 DATA13,222,17,192,189,17,192,201
2360 DATA255,208,3,222,18,192,189,17
2370 DATA192,221,4,192,189,18,192,253,5
2380 DATA192,144,18,189,4,192,157,17
2390 DATA192,189,5,192,157,18,192,104
2400 DATA74,72,76,46,193,104,74,72,176
2410 DATA8,254,17,192,208,3,254,18,192
2420 DATA104,74,176,5,169,1,153,25,192
2430 DATA208,232,232,224,4,240,3,76,91
2440 DATA192,238,27,192,108,29,192
2450 DATA35994:REM*CHECKSUM*
2460 FORI=49183TO49478
2470 READX:POKEI,X:CC=CC+X
2480 NEXT
2490 READX:IFX<>CCTHEN PRINT"CHECKSUM ERROR."
2500 RETURN

```

Once the subroutine is RUN, we have several other things to do before we can use the joystick. The first thing to do is to set the upper and lower limits of X and Y. This is done by POKEing numbers into these locations.

Limit	Joystick	Lo-Byte	Hi-Byte
X MIN	2	49152	49153
X MIN	1	49154	49155
X MAX	2	49156	49157
X MAX	1	49158	49159
Y MIN	2	49160	49161
Y MIN	1	49162	49163
Y MAX	2	49164	49165
Y MAX	1	49166	49167

As you can see, each limit has two locations. One is known as the 'Lo-Byte' and the other the 'Hi-Byte'. The largest number that can be held in any location is 255 (11111111 in binary). However, visible values of X range from 24 to 343, so a second location is required to hold values of X greater than 255.

The Lo-Byte and Hi-Byte combine like this:

Value of X	Hi-Byte	Lo-Byte																
254	<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>	0	0	0	0	0	0	0	0	<table><tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td></tr></table>	1	1	1	1	1	1	1	0
0	0	0	0	0	0	0	0											
1	1	1	1	1	1	1	0											
255	<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>	0	0	0	0	0	0	0	0	<table><tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr></table>	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0											
1	1	1	1	1	1	1	1											
256	<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td></tr></table>	0	0	0	0	0	0	0	1	<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1											
0	0	0	0	0	0	0	0											
257	<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td></tr></table>	0	0	0	0	0	0	0	1	<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td></tr></table>	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1											
0	0	0	0	0	0	0	1											

An upper limit to X may be set to 321 for joystick 2 by this POKE:

POKE49156,65:POKE49157,1

The next task is to set the scan rate. This is simply the rate at which the joystick is read. POKEing location 49168 with a number in the range 0 - 255, this will set the scan rate between 1/60th and 4.3 seconds respectively.

Before your program enters its main 'reading loop' the last thing to be done is to enter the machine-code program. This is known as entering the wedge and is accomplished by the following command:

SYS49183

The following locations should be PEKed to obtain values of X and Y:

Coord	Lo-Byte	Hi-Byte
X2	49169	49170
X1	49171	49172
Y2	49173	49174
Y1	49175	49176

The fire button registers have addresses 49177, for fire button 2, and 49178 for fire button 1. When PEEKing these locations within a loop use the following routine:

```

100 POKE49177,0:POKE49178,0
110 FB2=PEEK(49177):FB1=PEEK(49178)
120 IF FB1=1 THEN <action>
130 IF FB2=1 THEN <action>
140 GOTO 100

```

Both fire button registers should be reset at some stage in the loop, this is probably best done just prior to PEEKing them.

At the end of a program you should disable the wedge using SYS49208.

If you break out of a program that uses the machine-code joystick routine you may find that when you try to RUN it again the joystick will not appear to work. To get round this problem, before you RUN the program again press RUN/STOP and RESTORE. Alternatively, type in SYS49208 to disable the wedge.

To tie all these ends together here is a program that uses the machine-code joystick subroutine.

```

1 REM  ****
2 REM  ****
3 REM  **
4 REM  **      M/C      **
5 REM  ** JOYSTICK **
6 REM  **      DEMO      **
7 REM  **
8 REM  ****
9 REM  ****
10 PRINTCHR$(147):REM CLEAR SCREEN
20 V=53248
30 GOSUB2000:REM M/C JOYSTICK
100 REM-----SET LIMITS-----
110 POKE49152,24:POKE49153,0: REM X2 MIN
120 POKE49156,65:POKE49157,1: REM X2 MAX
130 POKE49160,50:POKE49161,0: REM Y2 MIN
140 POKE49164,229:POKE49165,0:REM Y2 MAX
150 REM-----SET SCAN RATE-----

```

Listing continued next page

```

100 X2=141:Y2=120
120 REM---CREATE SPRITE-----
210 FORI=1:208TO12350:POKEI,255:NEXT
220 POKE3349,192:REM MEN POINTERS
230 POKEV+39,7:REM COLOR
240 POKEV+21,1:REM TURN ON
250 REM-----SET INITIAL SPR POSITION-----
260 POKE49169,168:POKE49170,0:REM X2
270 POKE49173,148:POKE49174,0:REM Y2
300 REM-----ENTER WEDGE-----
320 SYS49108
330 REM-----MAIN PROGRAM LOOP-----
310 X2=PEEK(49169)+256*PEEK(49170)
320 Y2=PEEK(49173)+256*PEEK(49174)
330 REM---OPEN PHS OF SCREEN---
340 H2=INT(X2/256):L2=X2-256*H2
350 POKEV+16,H2
360 REM---POSITION SPRITE---
370 POKEV,L2:POKEV+1,Y2
380 REM---FIRE BUTTON---
390 POKE49177,0:REM RESET FB REGISTER
400 FB3=PEEK(49177)
410 IFFB3<1THEN300
420 PRINTCHR$(145):REM CURSOR UP
430 PRINTTAB(13)"BOMBS AWAY"
440 FORI=1TO500:NEXT
450 PRINTCHR$(147):REM CLEAR SCREEN
460 GOTO300
2000 REM-----B/L FOR N/C JOYSTICK-----
2010 DATA120,173,20,3,141,29,192,173,21
2020 DATA3,141,30,192,169,71,141,20,3
2030 DATA169,192,141,21,3,89,96,120,173
2040 DATA29,192,141,20,3,173,30,192,141
2050 DATA21,3,89,96,173,27,192,205,16
2060 DATA192,240,3,76,65,193,169,255
2070 DATA141,27,192,162,0,160,0,185,2
2080 DATA228,141,20,192,169,224,153,2

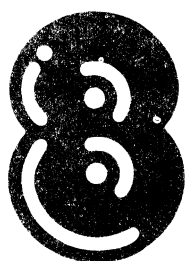
```

```

2090 DATA220,185,0,220,72,173,28,192
2100 DATA153,2,220,189,8,192,221,21,192
2110 DATA189,9,192,253,22,192,144,18
2120 DATA189,8,192,157,21,192,189,9,192
2130 DATA157,22,192,104,74,72,76,162
2140 DATA192,104,74,72,176,13,222,21
2150 DATA192,189,21,192,201,255,208,3
2160 DATA222,22,192,189,21,192,221,12
2170 DATA192,189,22,192,253,13,192,144
2180 DATA18,189,12,192,157,21,192,189
2190 DATA13,192,157,22,192,104,74,72,76
2200 DATA207,192,104,74,72,176,8,254,21
2210 DATA192,208,3,254,22,192,189,0,192
2220 DATA221,17,192,189,1,192,253,18
2230 DATA192,144,18,189,0,192,157,17
2240 DATA192,189,1,192,157,18,192,104
2250 DATA74,72,76,1,193,104,74,72,176
2260 DATA13,222,17,192,189,17,192,201
2270 DATA255,208,3,222,18,192,189,17
2280 DATA192,221,4,192,189,18,192,253,5
2290 DATA192,144,18,189,4,192,157,17
2300 DATA192,189,5,192,157,18,192,104
2310 DATA74,72,76,46,193,104,74,72,176
2320 DATA8,254,17,192,208,3,254,18,192
2330 DATA104,74,176,5,169,1,153,25,192
2340 DATA200,232,232,224,4,240,3,76,91
2350 DATA192,238,27,192,108,29,192
2360 DATA35994:REM*CHECKSUM*
2370 FORI=49183TO49478
2380 READX:POKEI,X:CC=CC+X
2390 NEXT
2400 READX:IFX<>CCTHEN PRINT"CHECKSUM ERROR."
2410 RETURN

```

A full explanation of how this machine-code routine was written is included in *Mastering the CBM 64* by A. J. Jones and G. Carpenter, also published by Ellis Horwood.



Beginner's sound advice

INTRODUCTION

The Sound Interface Device or 'SID' chip, is one of the most advanced sound synthesizers to be found on any home computer. You can, with a lot of effort create almost any sound. Unfortunately, user access is not very friendly. A series of POKEs are required and this can lead to utter confusion if you're not careful.

In this chapter we will try to give you an insight into the basic workings of the 'SID' chip.

SID has three oscillators and each has a range of seven octaves (refer to Appendix M of the User Manual for the table of notes); also each voice has an ADSR capability (ATTACK, SUSTAIN, DECAY, RELEASE), complete with four different waveforms. These are, Triangle, Sawtooth, Square (or Pulse) and White Noise.

We will start with volume control. This is controlled at location 54296 and only the first four bits of the register are used for the volume control, which can be set in the range 0-15.

Secondly we have the Waveform, and our choice out of the four available, will determine the timbre or tonal quality of the sound. This can best be described by Fig. 8.1.

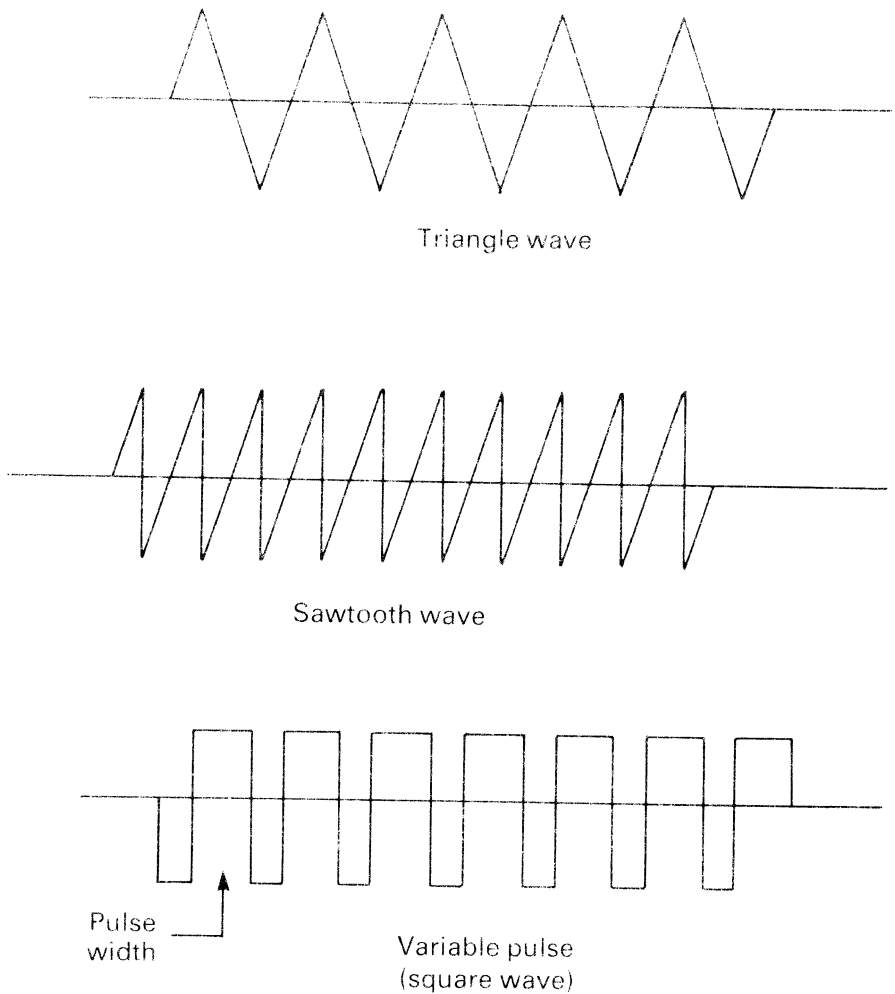


Fig. 8.1

Thirdly we come to the ADSR feature. 'ATTACK' determines the rate at which the note rises to its peak. 'DECAY' is the rate at which the note falls to zero. 'SUSTAIN' is the time for which the note holds and 'RELEASE' lets you extend a particular note (see Fig. 8.2).

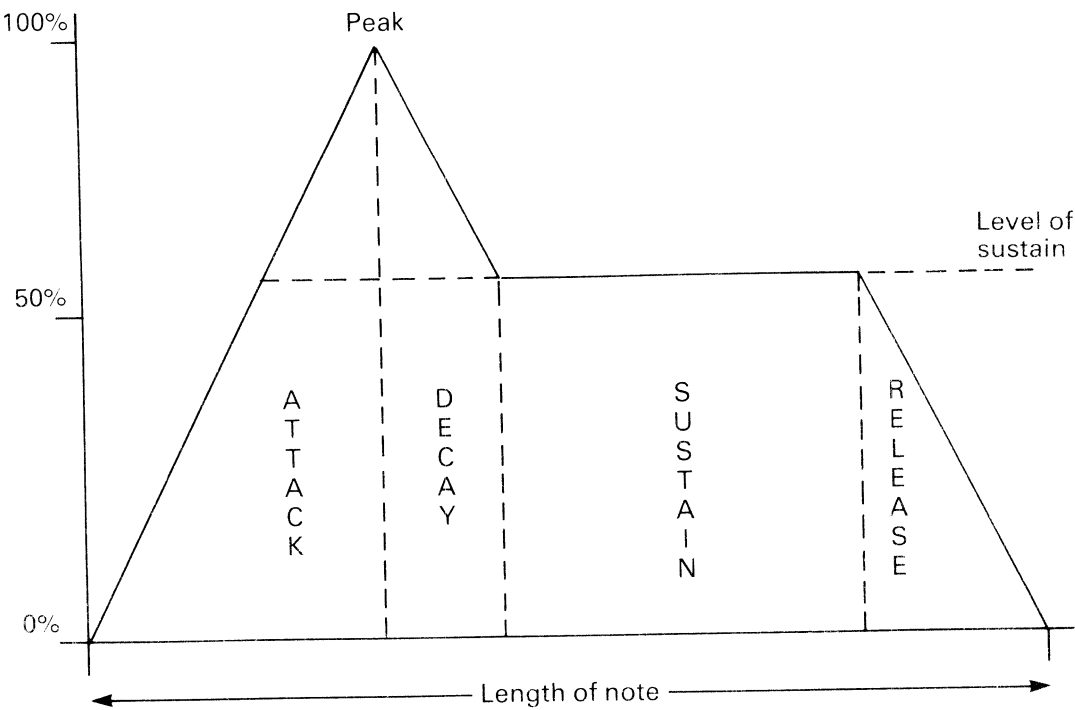


Fig. 8.2

Lastly, and perhaps the most important, are the tone generators themselves. These are locations which hold the frequency of a note. The first one is the high frequency and is held at locations 54273 for voice 1, 54280 for voice 2 and 54287 for voice 3. The low frequency is held at locations 54272 for voice 1, 54279 for voice 2 and 54286 for voice 3.

The main principles of producing a sound are:

- | | |
|--|-------------------|
| (a) SET VOLUME | (1-15) |
| (b) SET WAVEFORM | (17, 33, 65, 129) |
| (c) SET ATTACK/DECAY | (0-255) |
| (d) SET SUSTAIN/RELEASE | (0-255) |
| (e) SET HIGH PULSE (if using pulse waveform) | (0-15) |
| (f) SET LOW PULSE (if using pulse waveform) | (0-255) |
| (g) SET HIGH FREQUENCY | (0-255) |
| (h) SET LOW FREQUENCY | (0-255) |

After a note has been played you must clear the ADSR or you will get odd things happening, like a quiet note which continues to sound or, the next time the note is played, not sounding very loud. You may have come across this before.

O.K. now that we have whetted your appetite, we will put a little of what we have learned into practice. The first demo will play a note determined in length by a delay loop. You will notice that a lot has to be done even for a very simple note. Type in this program:

```

1010 FORI=0TO24 :REM -----
1020 POKES+1,0 :REM Clear 'SID' chip
1030 NEXTI :REM -----
1040 POKES+24,15 :REM Set volume control
1050 POKES+4,33 :REM Set up Sawtooth Waveform
1060 POKES+5,31 :REM Set up Attack/Decay
1070 POKES+6,15 :REM Set up Sustain/Release
1080 POKES+1,51 :REM Set up Hi-Frequency
1090 POKES+0,97 :REM Set up Lo-Frequency
1100 FORI=1TO500 :REM -----
1110 NEXTI :REM Delay loop
1120 POKES+4,0 :REM Clear Waveform
1130 POKES+5,0 :REM Clear Attack/Decay
1140 POKES+6,0 :REM Clear Sustain/Release

```

After you have typed in this program, proof-read it and then RUN. You should hear a note.

If you are using sound effects in a program, set up most of the parameters before you get into the main loop like this:

```

1000 S=54272      :REM start of 'SID' chip
1010 FOR I=0 TO 24 :REM -----
1020 POKE S+I,0    :REM clear 'SID' chip
1030 NEXT I        :REM -----
1040 POKE S+5,31   :REM set up ATTACK/DECAY
1050 POKE S+6,15   :REM set up SUSTAIN/RELEASE
2000 -----
2010 MAIN CONTROL LOOP OF PROGRAM
2020 -----
3000 REM SUBROUTINE FOR SOUND
3010 POKE S+4,33   :REM SAWTOOTH wave
3020 POKE S+1,25   :REM HI-FREQUENCY
3030 POKE S+0,90   :REM LO-FREQUENCY
3040 FOR I=1 TO 500 :REM DELAY
3050 NEXT I        :REM LOOP
3060 POKE S+4,0    :REM clear WAVEFORM

```

It lies beyond the scope of this book to delve any further into the intricacies of sound programming, but here are a few sound effects for you to try. Use them in your own programs or analyse them to help you devise your own sounds. Appendix P in your User Manual has some of the key numbers required to produce those scintillating sounds for your 64!

EXAMPLE PROGRAMS

```

1000 REM *** FX 1 (EXPLOSION) ***
1010 S=54272
1020 FORI=0TO24
1030 POKES+I,0
1040 NEXTI
1050 POKES+5,31
1060 POKES+6,240
1070 POKES+4,129
1080 FORV=15TO0STEP-.2
1090 POKES+24,V
1100 POKES+1,2
1110 POKES+0,90
1120 NEXTV
1130 POKES+4,0

```

```

1000 REM *** FX 2 (UFO) ***
1010 S=54272
1020 FORI=0TO24
1030 POKES+I,0
1040 NEXTI
1050 POKES+24,15
1060 A=1
1070 POKES+5,240
1080 POKES+6,190
1090 POKES+4,17
1100 FORT=1TO100
1110 FORI=0TO50STEPA
1120 POKES+1,16+I+A
1130 POKES+0,200
1140 NEXTI
1150 A=A+.1
1160 IF A>49 THEN A=1
1170 NEXTT
1180 POKES+4,0

```

```

1000 REM *** FX 3 (RED ALERT) ***
1010 S=54272
1020 FORI=0TO24
1030 POKES+I,0
1040 NEXTI
1050 POKES+24,15
1060 POKES+5,240
1070 POKES+6,199
1080 POKES+4,33
1090 PORT=1TO20
1100 FORI=0TO50STEP1
1110 POKES+1,16+I
1120 POKES+0,200
1130 NEXTI,1
1140 POKES+4,0

```

```

1000 REM *** FX 4 (FROG) ***
1010 S=54272
1020 FORI=0TO24
1030 POKES+I,0
1040 NEXTI
1050 POKES+24,15
1060 POKES+5,47
1070 POKES+6,31
1080 FORI=0TO80STEP9
1090 POKES+4,0
1100 POKES+1,I
1110 POKES+0,20
1120 POKES+4,17
1130 NEXTI
1140 POKES+4,0

```



Advanced tips and hints

SOME 'DO'S AND 'DON'T'S

- DO make your action loop as short as possible, preferably less than one full screen for fast-moving programs.
- DO use REMs at the beginning of each program section and when using GOSUB.
- DO speed up your programs by using variables and constants, instead of numbers.
- DO think out your program in blocks and plan out a rough structure before starting.
- DO make good use of subroutines.
- DO save your program BEFORE attempting a run, especially if the program is incomplete.
- DO use PRINT rather than POKE if you want to move characters around the screen. It's less elegant but it's quicker!
- DO tell your friends about this book.

- DON'T jump out of subroutines without going through RETURN.
- DON'T jump out of a FOR . . . NEXT loop without going through the last NEXT.
- DON'T always use the first solution that comes to mind when presented with a problem. Think it through; there's probably a quicker and/or neater way.
- DON'T forget to tell your friends about this book!

SOME INTERESTING POKES ETC.

To disable the LIST command type:

POKE 775,200

The following POKE enables the LIST command:

POKE 775,167

To disable the RUN/STOP key type:

POKE 808,239

The RUN/STOP key can be re-enabled by:

POKE 808,237

Disabling RUN/STOP and RESTORE is accomplished by the following POKE:

POKE 808,235

The function is re-enabled by:

POKE 808,235

The keyboard is disabled by:

POKE 649,0

and re-enabled by:

POKE 649,10

The screen can be blanked by:

POKE 53265,PEEK(53265) AND 239

and restored to its former glory by:

POKE 53265,PEEK(53265) OR 16

To print a character at column X and row Y:

```
POKE 782,X
POKE 781,Y
SYS 65520
```

X should be in the range 0–39 and Y in the range 0–24.

FURTHER EXCITING SNIPPETS

The Commodore 64 is ‘cold-started’ by SYS 64738. This command resets the machine to its original state.

PRINT FRE(0)–(FRE(0)<0)*65536 gives you the current number of BASIC bytes free.

The screen can be scrolled DOWN by the following sequence of commands:

```
PRINT CHR$(19);CHR$(17);CHR$(157);CHR$(148)
POKE 218,160
```

Try changing the POKE to POKE 218,158 and see what happens!

THE COLLISION REGISTERS

Always clear a collision register before you PEEK it. This is done by POKEing V+30 (or V+31) with 0. The collision registers are cleared automatically but this does not always happen quickly enough (see Chapter 6). Remember that V = 53248.

The following POKE will turn off any two sprites involved in a collision:

```
POKE V+21,PEEK(V+21)AND(255-PEEK(V+30))
```

THE USE OF THE RETURN KEY IN EDITING

A quick way to abort a line in direct mode is to press the SHIFT and RETURN keys together. The line will then not be executed.

If you run into difficulties whilst editing inside quotation marks, press SHIFT/RETURN then reposition the cursor and complete the edit.

TROUBLE-SHOOTING

All the programs in this book have been thoroughly tested and will work correctly if entered as LISTed. If you have problems getting a game to work check the LISTing carefully. It is much easier for two people to do this: get a friend to read the LISTing whilst you check it against the CBM-64 screen. The most likely error message to appear is 'SYNTAX ERROR IN LINE XXX', in which case you have probably made a simple typing error (by far the most common is a missing bracket!). If there are no syntax errors and things are still not going well it may be that you have missed out a DATA statement, here the error message would be 'OUT OF DATA ERROR'. All the long blocks of DATA statements have been provided with checksums, i.e. the value obtained by adding all the DATA statements together, and you should get this right before proceeding further.

It may seem obvious but nevertheless it is worth saying: after entering a program ALWAYS SAVE it BEFORE RUNning it. This is especially true of any program with a machine-code subroutine.

UTILITY AND DEMONSTRATION PROGRAMS

Here are some programs that you may find of use when programming your own games on the Commodore 64. Particularly useful is the 'Sprite Generator'. Other programs include demonstrations of string-handling facilities.

Sprite Generator

This generator was written solely for the purpose of designing sprites and, once designed, giving you the DATA statements for that character.

Type in the 'Sprite Generator' program and RUN it. The left-hand side of the screen displays a box in which you will see a flashing cursor. This can be moved around the box, wrapping around horizontally and vertically at the edges. The cursor is moved with the aid of a joystick.

There are two modes, one for INSERT and another for DELETE. When you are in INSERT mode, every time you press the fire button a block appears under the cursor, and in DELETE mode any block under the cursor disappears. Each mode is represented by a different border color, 'GREEN' for INSERT mode and 'RED' for DELETE mode.

As you create your character in the large box, so a sprite appears in the small box on the right-hand side of the screen. If you delete a block, the same pixel on the Sprite will be turned off. This sprite is in fact expanded, both horizontally and vertically. If at any time you are not satisfied with your creation, then simply press the 'CLR/HOME' key and both boxes will be cleared.

Once you have finished press the 'F1' key to display the DATA for the sprite you have created. (Make sure you have no ROM-based cartridge installed that utilizes this key or you will get strange results.) Copy down the DATA and press the fire button to start a new design.

A word of warning. Don't ask for the data to be displayed until you are happy with your sprite as, in this program, there is no way of re-editing once the data statements have been displayed!

This 'Sprite Generator' program proved invaluable in the writing of this book; we hope you have as much fun with it as we have had.

```

1000 REM **** CBM 64 SPRITE GENERATOR ****
1010 PRINTCHR$(147)
1020 POKE53280,5:POKE53281,0
1030 S$="XXXXXXXXXXXXXXXXXXXX"
1040 D$="XXXXXXXXXXXXXXXXXXXX"
1050 FORI=0TO7:CY$(I)=2↑(7-I):NEXT
1060 SC=1105:CO=55377
1070 GOSUB1450:REM SCREEN DISPLAY
1080 REM ***** MAIN LOOP *****
1090 K=PEEK(197):IFK=51THENGOSUB1450
1100 IFK=4THENGOSUB1270
1110 IFK=33THENF=0:REM 'INSERT' FLAG
1120 IFK=18THENF=1:REM 'DELETE' FLAG
1130 POKESC+40*Y+X,M1
1140 P=PEEK(56320):REM JOYPORT #2
1150 IFP=126THENY=Y-1:IFY<0THENY=20
1160 IFP=125THENY=Y+1:IFY>20THENY=0
1170 IFP=123THENX=X-1:IFX<0THENX=23
1180 IFP=119THENX=X+1:IFX>23THENX=0
1190 IFF=0ANDP=111THENPOKESC+X+40*Y,160:GOSUB1840
1200 IFF=1ANDP=111THENPOKESC+X+40*Y,79:GOSUB1890
1210 IFF=0THENPOKE53280,5:REM GRN BORDER
1220 IFF=1THENPOKE53280,2:REM RED BORDER
1230 M2=PEEK(SC+X+40*Y):M1=M2
1240 POKECO+X+40*Y,14:POKESC+X+40*Y,81
1250 FORL=1TO50:NEXT
1260 GOTO1090
1270 REM ***** DATA STATEMENTS *****
1280 PRINTCHR$(147):FORL=1TO3
1290 PRINTCHR$(17):NEXT

```

Listing continued next page

```

1300 PRINT"      _____SPRITE. _____";
1310 PRINTCHR$(18)"▀"CHR$(146)
1320 PRINT"      _____▀"
1330 PRINT:PRINT:PRINT
1340 FORD=0T08
1350 PRINT" DATA";
1360 FORI=832+(D*7)T0838+(D*7):P=PEEK(I)
1370 PRINTP;CHR$(157)CHR$(44)" ";
1380 PRINTCHR$(157);:NEXT:PRINTCHR$(157)" ":NEXT
1390 PRINT
1400 PRINTCHR$(18)TAB(6)"▀PRESS FIRE ";
1410 PRINT" TO CREATE SPRITE▀"
1420 IFPEEK(56320)<>1111THEN1420
1430 GOSUB1450:RETURN
1440 REM ***** MAIN DISPLAY *****
1450 M1=79
1460 PRINTCHR$(147)CHR$(30)CHR$(18)" *** CBM ";
1470 PRINT"64 - SPRITE GENERATOR *** ";
1480 PRINTCHR$(146)CHR$(154)" _____";
1490 PRINT" _____"
1500 FORT=1T021
1510 PRINTCHR$(18)"▀"CHR$(146)"TTTTTT";
1520 PRINT"TTTTTTTTTTTTTTTTTT▀"
1530 NEXT
1540 PRINT" "CHR$(18)" _____";
1550 PRINT" _____"CHR$(146)" "
1560 PRINTCHR$(30)CHR$(18)" [ PRESS ";
1570 PRINT"FUNCTION KEY 1 FOR ";
1580 PRINT"DATA ] "CHR$(19)
1590 X=0:Y=0:POKE56295,5:POKE2023,160
1600 PRINTS$;D$;CHR$(18)"I"CHR$(146)"=INSERT MODE"
1610 PRINTS$;D$;RIGHT$(S$,1)" (GRN BORDER)"
1620 PRINTS$;D$;RIGHT$(S$,2);CHR$(18)"D"CHR$(146)"=DELE
TE MODE"
1630 PRINTS$;D$;RIGHT$(S$,3)" (RED BORDER)"
1640 PRINTS$;D$;RIGHT$(S$,4);CHR$(18)"CLR"CHR$(146)"=NE
W "

```

```

1650 PRINTS$;D$;RIGHT$(S$,6)"TRIGGER TO"
1660 PRINTS$;D$;RIGHT$(S$,7)"PLOT & UNPLOT"
1670 K$=LEFT$(S$,3)+LEFT$(D$,26)
1680 PRINTK$;CHR$(145)" * SPRITE. *":PRINTK$;
1690 PRINT" _____"
1700 FORK=1TO11:PRINTLEFT$(D$,26);
1710 PRINT" | "":NEXT
1720 K$=LEFT$(S$,14)+LEFT$(D$,26)
1730 PRINTK$" L _____"
1740 VC=53248
1750 POKEVC+21,1:REM ENABLE SPRITE 0
1760 POKE2040,13:REM 13*64=832 PTR
1770 POKEVC,5:REM X COORD LO
1780 POKEVC+16,1:REM X COORD HI
1790 POKEVC+1,95:REM Y COORD
1800 POKEVC+39,5:REM GREEN
1810 POKEVC+29,1:POKEVC+23,1:REM EXPAND
1820 FORI=0TO62:POKE832+I,0:NEXT
1830 RETURN
1840 REM ** INSERT PIXEL IN SPRITE **
1850 TH=INT(X/8):BI=X-8*TH
1860 BY=832+3*Y+TH
1870 POKEBY,PEEK(BY)ORC%(BI)
1880 RETURN
1890 REM ** DELETE PIXEL IN SPRITE **
1900 TH=INT(X/8):BI=X-8*TH
1910 BY=832+3*Y+TH
1920 POKEBY,PEEK(BY)AND(255-C%(BI))
1930 RETURN

```

Here is a program to change the normal character set into space age characters.

```

1000 REM***SPACE CHARSET***
1010 REM *LOWER MEMTOP*
1020 POKE52,48:POKE56,48:CLR
1030 FORI=0TO511
1040 POKEI+12288,PEEK(I+53248):NEXT
1050 REM *ALPHABET*
1060 FORALPHA=12288TO12503:READX:AA=AA+X:POKEALPHA,X:NEXTALPHA
1070 READX:IFX<>A THENPRINT"DATA ERROR":END
1080 REM *NUMBERS*
1090 FORNUM=12672TO12751:READZ:BB=BB+Z:POKENUM,Z:NEXTNUM
1100 READZ:IFZ<>B THENPRINT"DATA ERROR":END
1110 REM *POINTER*
1120 POKE53272,(PEEK(53272)AND240)+12
1130 REM*ALPHABET DATA*
1140 DATA0,0,0,0,0,0,0,0
1150 DATA252,132,132,254,134,134,134,0
1160 DATA252,132,132,254,134,134,254,0
1170 DATA252,128,128,128,128,128,254,0
1180 DATA240,136,132,134,134,134,254,0
1190 DATA248,128,128,252,128,128,254,0
1200 DATA254,128,128,252,128,128,128,0
1210 DATA252,128,128,134,134,134,254,0
1220 DATA132,132,132,254,134,134,134,0
1230 DATA16,16,16,56,56,56,56,0
1240 DATA254,16,16,24,24,24,248,0
1250 DATA128,132,136,240,140,140,140,0
1260 DATA128,128,128,128,128,128,254,0
1270 DATA204,180,132,134,134,134,134,0
1280 DATA132,132,228,158,134,134,134,0
1290 DATA252,132,132,134,134,134,254,0
1300 DATA254,130,254,128,128,128,128,0
1310 DATA252,132,132,150,142,134,254,1
1320 DATA254,130,252,134,134,134,134,0
1330 DATA248,128,252,6,6,6,254,0
1340 DATA254,16,16,24,24,24,24,0
1350 DATA132,132,132,134,134,134,254,0

```

```

1360 DATA132,132,196,70,102,38,62,0
1370 DATA132,132,132,150,150,150,254,0
1380 DATA68,68,40,16,40,68,68,0
1390 DATA132,68,52,14,6,6,6,0
1400 DATA126,2,30,32,64,128,254,0
1410 REM *CHECKSUM*
1420 DATA24607
1430 REM *NUMBER DATA*
1440 DATA124,68,76,86,102,70,126,0
1450 DATA16,16,16,24,24,24,24,0
1460 DATA254,2,2,2,126,64,126,0
1470 DATA28,4,4,62,6,6,254,0
1480 DATA132,132,132,254,6,6,6,0
1490 DATA252,128,128,128,248,8,248,0
1500 DATA128,128,128,252,132,132,252,0
1510 DATA252,4,4,6,6,6,6,0
1520 DATA120,72,252,134,134,134,254,0
1530 DATA252,132,252,6,6,6,6,0
1540 REM *CHECKSUM*
1550 DATA6740

```

```

1000 REM *** STRING DEMO 1 ***
1010 PRINTCHR$(147)CHR$(30)
1020 POKE53280,0:POKE53281,0
1030 A$(1)="ATTACK COMPUTER ON"
1040 A$(2)="PHASERS LOCKED ON"
1050 A$(3)="TARGET...."
1060 A$(4)="SHEILDS ON MAXIMUM"
1070 A$(5)="ENERGY SUPPLY 10.547"
1080 A$(6)="ENGINES AT WARP .003"
1090 A$(7)="ENEMY IN SECTOR"
1100 A$(8)="CURRENT STATUS NOMINAL"

```

Listing continued next page

```

1110 A$(9)="DIRECT HIT..."
1120 A$(10)="AWAIT COMMAND INPUT"
1130 FORL=1TO10
1140 FORT=1TO22
1150 FORA=1TO30:NEXT
1160 PRINTLEFT$(A$(L),T)
1170 PRINTCHR$(145)CHR$(145):NEXT
1180 PRINT:PRINT:NEXT

```

```

1000 REM *** STRING DEMO 2 ***
1010 PRINTCHR$(147)
1020 FORX=17TO1STEP-1
1030 READA$
1040 IFA$="*"THEN1100
1050 IFA$="@"THEN:GOSUB1180:END
1060 FORL=0TOX
1070 PRINTTAB(L+9)" "A$
1080 PRINTCHR$(145)CHR$(145)
1090 NEXTL,X
1100 PRINT:PRINT:GOSUB1180
1110 GOTO1020
1120 DATA , ,E,R,E,H, ,D,R,O,W, ,Y,N,A,*
1130 DATA , ,E,H,T, ,N,I, ,D,E,R,O,T,S,*
1140 DATA ,S,T,N,E,N,E,T,A,T,S, ,A,T,A,D,*
1150 DATA , ,N,O, ,T,U,P, ,E,B, ,N,A,C,*
1160 DATA , ,N,E,E,R,C,S, , ,E,H,T, ,*
1170 DATA , , ,S,I,H,T, ,E,K,I,L, , ,@
1180 FORT=1TO50:NEXT:RETURN

```

```
1000 REM *** STRING DEMO 3 ***
1010 PRINTCHR$(147)
1020 PRINTTAB(10)" _____"
1030 PRINTTAB(10)" |"
1040 PRINTTAB(10)" L _____"
1050 A$="  COMMODORE * 64  "
1060 PRINTCHR$(19)CHR$(17)
1070 C$=RIGHT$(A$,17)+LEFT$(A$,1)
1080 A$=C$
1090 PRINTTAB(11);A$
1100 FORI=1TO50:NEXT
1110 GOTO1060
```


SECTION 2

Games listings

So much for the theory; this is what you have been waiting for. We hope that you have a lot of fun playing the games that we have written for you. We certainly had a lot of fun writing them. If you are typing them in yourself then be very careful to enter exactly what is given in the listing. Make sure that you save your program once it is typed in BEFORE you run it.

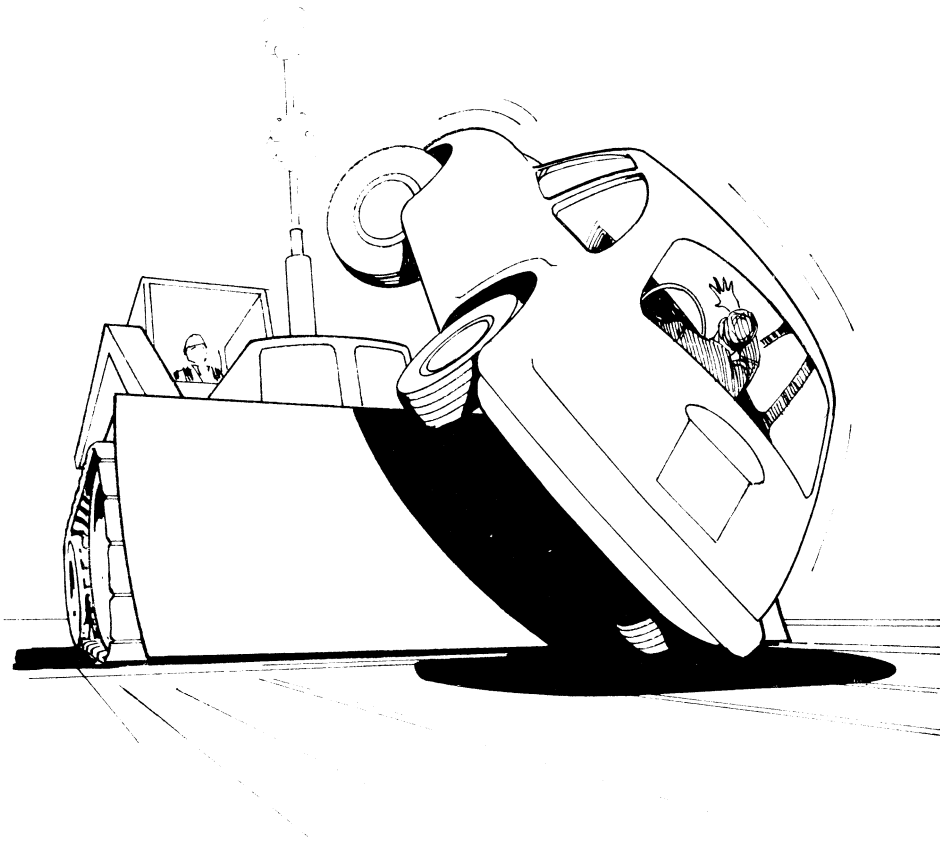
We have tried as far as possible to make our programs easy to read and they are structured in a way that you will find easy to follow and understand. We have avoided using the Commodore Control Characters for this reason, replacing them with the corresponding CHR\$. The only exceptions to this rule are the use in PRINT statements of RVS ON and RVS OFF control characters: the former is displayed as a reversed 'R', whilst the latter is a square with a horizontal line running across it. These are formed within quote mode by simultaneously depressing CTRL and 9 or CTRL and 0. All other oddities are the Commodore characters which are displayed on the front of each key. The right-hand character is obtained by pressing the appropriate key and the SHIFT key. The left-hand character is obtained by pressing the key together with the Commodore flag key.

Typing in long programs can be just a little tedious; but don't forget that all the programs listed in this section and all the demonstration programs used earlier are available on cassette or disk from the publishers.

Most of the programs that we have written require a joystick which should be plugged into CONTROL PORT 2, located on the right-hand side of the computer, next to the ON/OFF switch.

HAPPY GAMES PLAYING!

Demon Dozer



Demon Dozer will get you in the end; in fact if dozer doesn't a kerb will.

This is a game that requires a quick eye and lots of skill. When the game starts, a road will appear on the screen and a blue car, which is you, will be positioned in the centre of that road when suddenly, a dozer will come out of nowhere and try to ram you. You must evade the demon for as long as you can. Once your car starts to move it will keep on moving, making your task more difficult.

When you have played this game a few times, try altering the speed of the Demon Dozer or the speed at which you change direction.

```

1000 REM *** DEMON DOZER ***
1010 PRINTCHR$(147)
1020 L$="":FORI=1TO4:L$=L$+CHR$(157):NEXT
1030 POKE53280,0:POKE53281,0
1040 GOSUB1820:REM TITLE
1050 REM *** READ SPRITE DATA ***
1060 FORI=0TO62:READJ:POKE832+I,J:NEXT
1070 FORI=0TO62:READJ:POKE896+I,J:NEXT
1080 FORI=0TO62:READJ:POKE960+I,J:NEXT
1090 PRINTCHR$(147)
1100 REM *** SET UP SPRITES ***
1110 POKE2040,13:VC=53248:POKEVC+21,5
1120 POKE2041,14:POKE2042,15:POKEVC+40,8
1130 POKEVC+39,14:POKEVC+23,7:POKEVC+29,7
1140 POKEVC+0,150:POKEVC+1,180
1150 POKEVC+28,2:POKEVC+41,10
1160 REM *** SET UP SOUND ***
1170 S=54272:FORI=0TO24:POKES+I,0:NEXT
1180 S1=54286:POKES+5,31:POKES+6,240
1190 POKES+24,15:POKES1+5,31:POKES1+6,240
1200 L=12:X=150:W=12:Z=0:SE=0
1210 POKE646,14
1220 PRINTCHR$(19)CHR$(5)
1230 FORI=1TO21
1240 PRINTTAB(9)" " "SPC(16)" "
1250 NEXT:POKEVC+31,0:POKEVC+30,0
1260 POKES+4,33:POKES,100:POKES+1,4
1270 POKES1+4,33:POKES1,100:POKES1+1,5
1280 PRINTCHR$(19)CHR$(30)
1290 FORI=1TO21
1300 PRINTCHR$(18)" " "SPC(20);
1310 PRINT" ":REM BOTH 9 SPACES
1320 NEXTI
1330 PRINTCHR$(19)CHR$(5)"SCORE:"
1340 PRINTCHR$(19)TAB(28)"HIGH:"
1350 REM *** MAIN LOOP ***

```

Listing continued next page

```

1360 IFCR=0THENGOSUB1580
1370 SE=SE+1
1380 CY=CY+16:IFCY>210THENCY=1:CR=0
1390 P=PEEK(56320)
1400 IFP=123THENZ=1
1410 IFP=119THENZ=2
1420 ONZGOSUB1530,1550
1430 POKEVC+0,X
1440 POKEVC+4,R:POKEVC+5,CY
1450 PRINTCHR$(19)TAB(6)"      "L$SE
1460 IFSE>HITHENHI=SE
1470 PRINTCHR$(19)TAB(34);HI
1480 IFPEEK(VC+31)AND1=1THEN1600
1490 IFPEEK(VC+30)AND1=1THEN1600
1500 POKEVC+31,0:POKEVC+30,0
1510 GOTO1360
1520 REM *** CAR DIRECTION ***
1530 X=X-5:IFX<10THENX=10
1540 RETURN
1550 X=X+5:IFX>250THENX=249
1560 RETURN
1570 REM *** NEW POS OF DOZER ***
1580 R=INT(RND(TI)*84+110):CR=1:RETURN
1590 REM *** CRASH INTO KERB ***
1600 POKEVC+21,6:POKES+4,0
1610 POKEVC+2,X:POKEVC+3,180
1620 FORV=15TO0STEP-1:POKES+1,2+V
1630 POKE53270,INT(RND(1)*8)
1640 POKES+4,129:POKES+24,V
1650 POKEVC+37,INT(RND(TI)*15)
1660 POKEVC+38,INT(RND(TI)*15)
1670 NEXT:POKES+4,0
1680 POKEVC+21,0
1690 REM *** NEW GAME ***
1700 PRINTCHR$(19):CY=1:CR=0
1710 POKE646,INT(RND(TI)*15)

```

```

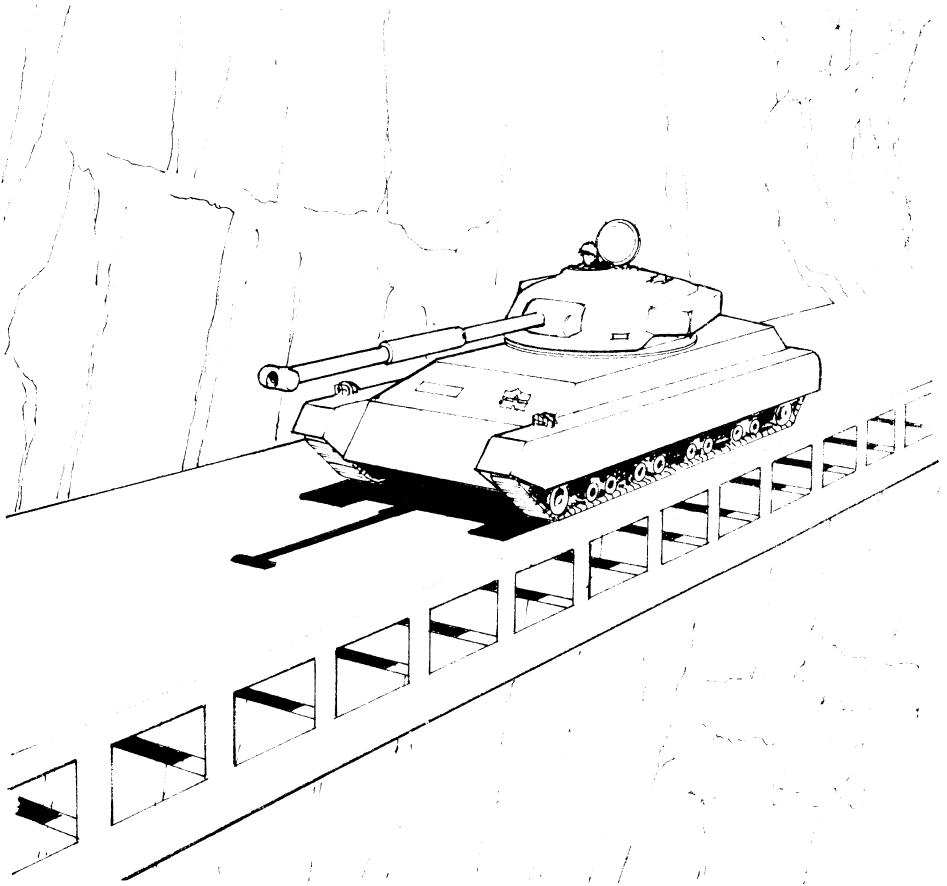
1720 FOR I=1 TO 4:PRINTCHR$(17):NEXT
1730 POKE53270,200
1740 PRINTTAB(13)"PRESS  FIRE"
1750 PRINTTAB(12)"TO PLAY AGAIN"
1760 POKEVC+4,R:POKEVC+5,CY
1770 IFPEEK(56320)<>1111THEN1700
1780 PRINTCHR$(145)CHR$(145)CHR$(145)
1790 PRINTTAB(13)"          ":REM 11
1800 PRINTTAB(12)"          ":REM 13
1810 GOTO1110
1820 REM *** TITLE PAGE ***
1830 PRINTCHR$(19)CHR$(5)
1840 PRINTTAB(7)"< D E M O N * ";
1850 PRINT"D O Z E R >"
1860 PRINTCHR$(17)CHR$(17)
1870 PRINTTAB(5)"USE JOYSTICK TO ";
1880 PRINT"STEER YOUR CAR"
1890 PRINTTAB(5)"AND AVOID THE ";
1900 PRINT"DEMON DOZER..."
1910 PRINTTAB(5)"ALSO WATCH OUT, ";
1920 PRINT"FOR THE KERB....."
1930 PRINTCHR$(17)CHR$(17)CHR$(17)
1940 PRINTTAB(9)"PRESS A KEY  TO PLAY"
1950 GETA$:IFA$=""THEN1950
1960 RETURN:REM INTO GAME
1970 REM *** SPRITE 0 (CAR) ***
1980 DATA0,0,0,7,129,224,7,189,224,7
1990 DATA126,224,6,255,96,6,255,96,7
2000 DATA255,224,4,0,32,6,0,96,5,255
2010 DATA160,5,255,160,5,255,160,7,255
2020 DATA224,5,255,160,5,255,160,6,0,96
2030 DATA4,0,32,7,255,224,7,255,224,7
2040 DATA255,224,6,255,96
2050 REM *** SPRITE 1 (EXPLODE) ***
2060 DATA0,32,0,2,73,52,6,157,193,104

```

Listing continued next page

```
2070 DATA188,29,131,54,124,44,118,249
2080 DATA65,231,242,31,227,178,78,8,57
2090 DATA39,63,156,47,159,30,73,222,124
2100 DATA36,194,240,73,248,198,19,253
2110 DATA216,71,205,144,23,15,36,16,103
2120 DATA72,13,146,80,0,8,144,3,87,32
2130 REM *** SPRITE 2 (DOZER) ***
2140 DATA0,10,0,12,10,48,13,255,176,15
2150 DATA255,240,12,255,48,13,189,176,1
2160 DATA219,128,1,255,128,5,0,160,7,36
2170 DATA224,7,129,224,5,255,160,1,255
2180 DATA128,0,129,0,0,129,0,7,255,224
2190 DATA8,0,16,0,0,0,0,0,0,0,0,0,0,0,0,0
```

3D Tank Assault



In this fantasy you play the part of an ambusher intent on destroying as many tanks as you can with the shells provided, and stopping them getting across the bridge to safety.

This game uses PET graphics to display the scenery quite effectively.

The tank and sight are sprites and, like most of the games in this book the sight is moved by utilizing the machine code joystick routine.

```

1000 REM *** 3D TANK ASSAULT ***
1010 PRINTCHR$(147):POKE53280,0:POKE53281,0
1020 POKE54296,15
1030 POKE53269,3:POKE2040,13:POKE2041,14
1040 FORI=0TO62:READJ:POKE832+I,J:NEXT:REM SIGHT
1050 FORI=0TO62:READJ:POKE896+I,J:NEXT:REM TANK
1060 FORI=49183TO49478:READJ:POKEI,J
1070 CC=CC+J:NEXTI:READJ
1080 IFJ<>CCTHENPRINT"M/C DATA ERROR":END
1090 SYS49183
1100 POKE53277,PEEK(53277)OR(211):REM EXPAND TANK
1110 POKE53250,250:POKE53251,150
1120 POKE53287,7:POKE53288,0:S=150:J=150
1130 SE=0:SH=99:R$=CHR$(18):S$=CHR$(19)
1140 SP$="" "":REM 20 SPACES
1150 GOSUB1630:REM DRAW SCREEN
1160 REM JOYSTICK PARAMETERS
1170 POKE49152,85:POKE49153,0
1180 POKE49156,250:POKE49157,0
1190 POKE49160,100:POKE49161,0
1200 POKE49164,200:POKE49165,0
1210 POKE49177,0:POKE49168,0
1220 POKE49169,150:POKE49173,100
1230 POKE53281,4
1240 REM MAIN LOOP
1250 T=T-4:IFT<40THENT=255
1260 POKE53250,T
1270 PRINT$:"SCORE:"SE:PRINT$;SPC(15)"HIGH:"HI
1280 PRINT$;SPC(29)"SHELLS:"SH:CHR$(157)" "
1290 IFSE>HITHENHI=SE
1300 X=PEEK(49169):Y=PEEK(49173)
1310 IFSH<1THEN1570
1320 POKE53248,X:POKE53249,Y
1330 F=PEEK(49177)
1340 POKE49177,0
1350 IFF=1THENSH=SH-1
1360 IFF=1ANDX=T+14ANDY=150THENGOSUB1390

```



```

1370 POKE49177,0
1380 GOTO1250
1390 REM TANK DISINTERGRATES
1400 POKE54296,15:SE=SE+1*SH:G=1
1410 SH=SH-1
1420 POKE54276,0:POKE54290,0
1430 POKE54276,33:POKE54290,33
1440 POKE54277,31:POKE54291,31
1450 POKE54278,240:POKE54292,240
1460 FORL=255TO18STEP-3
1470 POKE54273,L-7:POKE54287,L
1480 POKE54272,200:POKE54286,200
1490 POKE53288,L:NEXTL
1500 POKE54276,129:POKE54290,129
1510 FORV=15TO0STEP-1:POKE53296,3
1520 POKE54273,2:POKE54287,200+V
1530 POKE54272,90:POKE54286,90
1540 POKE54296,V:POKE53269,1:NEXT:T=255
1550 POKE54276,0:POKE54290,0:POKE53250,T
1560 POKE53288,0:POKE53269,3:RETURN
1570 REM *** END GAME ROUTINE ***
1580 PRINTCHR$(19)CHR$(17)CHR$(17)CHR$(5)
1590 PRINTTAB(7)"PRESS A KEY TO PLAY AGAIN"
1600 GETA$:IFA$=""THEN1600
1610 PRINTCHR$(147):POKE53281,0:SH=99:SE=0
1620 GOTO1150
1630 REM DRAW SCREEN (ALL SPACES =1
1640 PRINTCHR$(147):FORT=1TO5:PRINTCHR$(17):NEXT:
PRINTCHR$(144)
1650 PRINTSPC(39);R$ " ";
1660 PRINTSPC(37);R$;LEFT$(SP$,3);
1670 PRINTR$ " "SPC(33);R$;LEFT$(SP$,6);
1680 PRINTR$;LEFT$(SP$,2);CHR$(127);SPC(29);R$;CHR
$(190);LEFT$(SP$,7);
1690 PRINTR$;LEFT$(SP$,4);SPC(27);R$;LEFT$(SP$,9);

```

Listing continued next page

```

1700 PRINTR$;LEFT$(SP$,4);CHR$(127);SPC(25);R$;CHR
$(169);LEFT$(SP$,9);
1710 PRINTR$;LEFT$(SP$,6);CHR$(127);SPC(22)R$;CHR$
(191);LEFT$(SP$,10);
1720 PRINTR$;LEFT$(SP$,7);CHR$(172);SPC(20)R$;CHR$
(169)LEFT$(SP$,11);
1730 PRINTR$;LEFT$(SP$,9);:FORT=1T09
1740 PRINTCHR$(162);R$" ";:NEXT:PRINTR$;LEFT$(SP$,
12);
1750 PRINTR$;SP$;SP$;
1760 PRINTR$;LEFT$(SP$,10);CHR$(127);SPC(14);R$;CH
R$(169);LEFT$(SP$,14);
1770 PRINTR$;LEFT$(SP$,12);CHR$(188);SPC(9);R$;CHR
$(187);LEFT$(SP$,17);
1780 PRINTR$;LEFT$(SP$,12);CHR$(172);SPC(7)R$;SP$;

1790 PRINTR$;LEFT$(SP$,12);SPC(10);R$;CHR$(187);LE
FT$(SP$,17);
1800 PRINTR$;LEFT$(SP$,13);CHR$(172);SPC(5);R$;SP$
;" ";
1810 PRINTR$;LEFT$(SP$,15);CHR$(127);SPC(2);R$;CHR
$(169);SP$;" ";
1820 PRINTR$;SP$;SP$;
1830 PRINTR$;SP$;LEFT$(SP$,19);
1840 FORT=1T010;R=INT(RND(TI)*280):POKE55296+R,1:P
OKE1024+R,46:NEXT
1850 POKE2022,160:POKE56295,0
1860 POKE2023,160:POKE56294,0
1870 POKE55463,1:POKE1191,81
1880 RETURN
1890 REM SIGHT SPRITE
1900 DATA255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0
1910 DATA0,24,0,0,24,0,0,24,0,0,0,0,7,231,224,0,0,
0,0,24,0,0,24,0,0,24,0
1920 DATA0,0,0,0,0,0,0,0,0,0,0,0,255,255,255
1930 REM TANK SPRITE
1940 DATA0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0
1950 DATA0,8,0,0,31,0,0,63,128,63,255,192,0,127,19

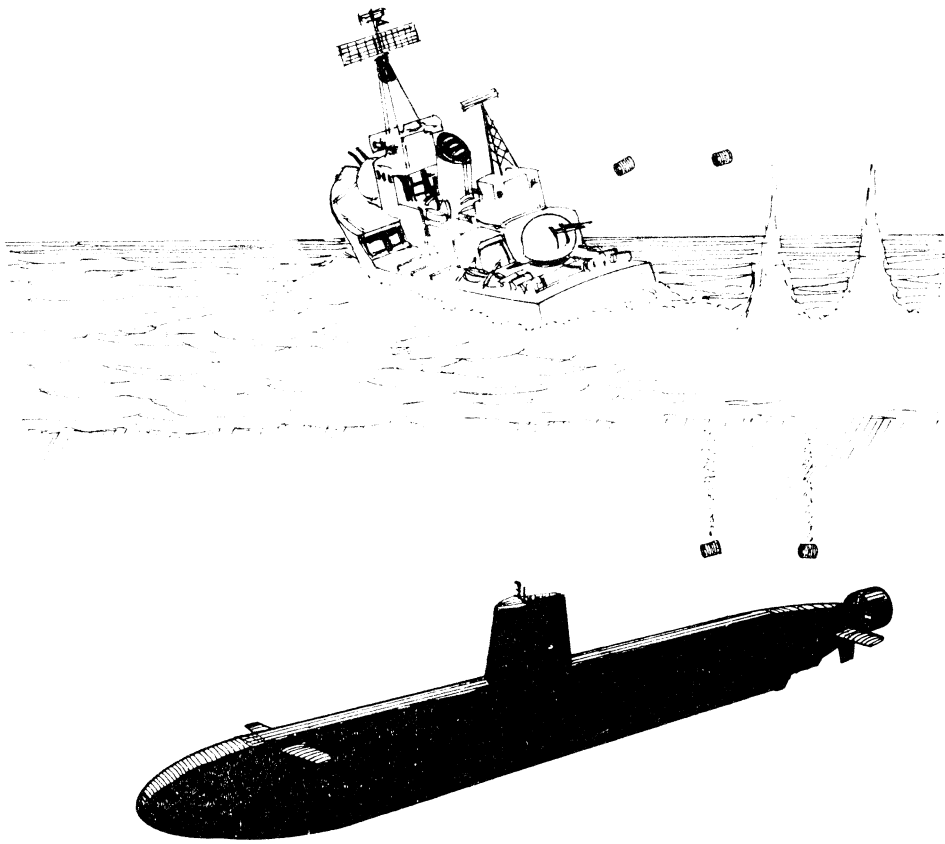
```

```

2,0,127,192
1960 DATA0,63,0,3,255,192,6,172,160,11,255,240
1970 DATA15,255,208,5,85,96,3,255,192
1980 REM M/C DATA
1990 DATA120,173,20,3,141,29,192,173,21
2000 DATA3,141,30,192,169,71,141,20,3
2010 DATA169,192,141,21,3,88,96,120,173
2020 DATA29,192,141,20,3,173,30,192,141
2030 DATA21,3,88,96,173,27,192,205,16
2040 DATA192,240,3,76,65,193,169,255
2050 DATA141,27,192,162,0,160,0,185,2
2060 DATA220,141,28,192,169,224,153,2
2070 DATA220,185,0,220,72,173,28,192
2080 DATA153,2,220,189,8,192,221,21,192
2090 DATA189,9,192,253,22,192,144,18
2100 DATA189,8,192,157,21,192,189,9,192
2110 DATA157,22,192,104,74,72,76,162
2120 DATA192,104,74,72,176,13,222,21
2130 DATA192,189,21,192,201,255,208,3
2140 DATA222,22,192,189,21,192,221,12
2150 DATA192,189,22,192,253,13,192,144
2160 DATA18,189,12,192,157,21,192,189
2170 DATA13,192,157,22,192,104,74,72,76
2180 DATA207,192,104,74,72,176,8,254,21
2190 DATA192,208,3,254,22,192,189,0,192
2200 DATA221,17,192,189,1,192,253,18
2210 DATA192,144,18,189,0,192,157,17
2220 DATA192,189,1,192,157,18,192,104
2230 DATA74,72,76,1,193,104,74,72,176
2240 DATA13,222,17,192,189,17,192,201
2250 DATA255,208,3,222,18,192,189,17
2260 DATA192,221,4,192,189,18,192,253,5
2270 DATA192,144,18,189,4,192,157,17
2280 DATA192,189,5,192,157,18,192,104
2290 DATA74,72,76,46,193,104,74,72,176
2300 DATA8,254,17,192,208,3,254,18,192
2310 DATA104,74,176,5,169,1,153,25,192
2320 DATA200,232,232,224,4,240,3,76,91
2330 DATA192,238,27,192,106,29,192
2340 DATA35994:REM *CHECKSUM*

```

Sub Hunt



'Radar room to bridge ... enemy subs gaining entry through underwater caverns ... drop depth charges!!!'.

The subs come at varying depths and speeds, so time your drop carefully or another one will escape. These submarines are quite clever too! Just when you think you've got one, it dives!!! Oh well, there's always another one.

Don't let more than 10 subs through or the game is over.

The ship is moved left and right across the screen by using the joystick and the charges are dropped by pressing the fire button.

Good hunting!

```

1000 REM *** SUB HUNT ***
1010 PRINTCHR$(147)
1020 POKE53280,0:POKE53281,6
1030 POKE54276,0:POKE54290,0
1040 FORI=49183TO49478:READJ
1050 POKEI,J:CC=CC+J:NEXTI
1060 READJ:IFCC<>JTHENPRINT"DATA ERROR":END
1070 FORI=0TO62:READJ:POKEI+832,J:NEXT
1080 FORI=0TO62:READJ:POKEI+896,J:NEXT
1090 FORI=0TO62:READJ:POKEI+960,J:NEXT
1100 SYS49183:REM INSERT JOYSTICK WEDGE
1110 SC=1024:CO=55296
1120 FORT=0TO239:POKECO+40+T,14
1130 POKESC+40+T,160:NEXTT
1140 FORT=0TO39:POKE56256+T,5
1150 POKE1984+T,102:NEXTT
1160 PRINTCHR$(19)CHR$(5)"SCORE:"SE
1170 PRINTCHR$(19)TAB(13)CHR$(18)CHR$(154)"<< SUB HUNT
>>"CHR$(146)
1180 PRINTCHR$(19)CHR$(5)TAB(30)"HIGH:"HI
1190 REM *** JOYSTICK PARAMETERS ***
1200 POKE49152,60:POKE49153,0
1210 POKE49156,255:POKE49157,0
1220 POKE49160,92:POKE49161,0
1230 POKE49164,92:POKE49165,0
1240 POKE49168,0:POKE49177,0
1250 POKE49169,150:POKE49173,100
1260 REM *** SET UP SPRITES ***
1270 VC=53248:PO=2040
1280 POKEPO,13:POKEPO+1,14
1290 POKEPO+2,15
1300 POKEVC+21,3:REM TURN ON SPRITE 0,1
1310 POKEVC+39,0:POKEVC+40,0:POKEVC+41,3
1320 POKEVC+29,3:REM EXPANDX 1,0
1330 POKEVC,50+INT(RND(TI)*150)
1340 POKEVC+2,150:POKEVC+3,92:REM SHIP

```

Listing continued next page

```

1350 POKEVC+4,120:POKEVC+5,170:REM BOMB
1360 POKEVC+39,0
1370 BY=110
1380 GOSUB1740
1390 REM *** MAIN LOOP ***
1400 A=PEEK(49169)
1410 IFSE>HITTHENHI=SE
1420 PRINTCHR$(19)TAB(6);SE:PRINTCHR$(19)TAB(35);HI
1430 POKEVC+2,X
1440 IFDR=0THENBX=X
1450 F=PEEK(49177)
1460 POKE49177,0
1470 IFF=1THENDR=1
1480 IFDR=1THENPOKEVC+21,7
1490 IFDR=1THENBY=BY+2
1500 IFBY>242THENBY=110:POKEVC+21,3:DR=0
1510 POKEVC+4,BX:POKEVC+5,BY
1520 POKEVC+30,0
1530 IFPEEK(VC+30)AND2=2THEN1580
1540 SX=SX+D:IFSX>254THENGOSUB1740:ESC=ESC+1:IFESC>9THE
N1790
1550 IFRND(1)<.03THENSY=SY+D:IFSY>220THENSY=220
1560 POKEVC+0,SX:POKEVC+1,SY
1570 GOTO1400
1580 REM *** DIRECT HIT ON SUB ***
1590 SE=SE+50+INT(SY/5)+D
1600 POKE54296,15:POKE54276,129:POKE54290,33
1610 POKE54277,31:POKE54291,31
1620 POKE54278,240:POKE54292,240
1630 POKEVC+21,3:BY=110:DR=0
1640 FORT=1TO50:R=INT(RND(TI)*15)
1650 POKEVC+39,R
1660 POKE54273,INT(RND(TI)*5)
1670 POKE54287,INT(RND(TI)*100)
1680 POKE54272,100:POKE54286,100
1690 NEXTT
1700 POKE54275,0:POKE54290,0

```

```

1710 POKEVC+21,2:FORT=1T0500:NEXT
1720 GOTO1360
1730 REM *** NEW SUB POSITION ***
1740 SY=150+INT(RND(TI)*75):SX=0
1750 POKEVC,SX:POKEVC+1,SX
1760 D=INT(RND(TI)*8+1)
1770 POKEVC+21,3:BY=110:DR=0
1780 RETURN
1790 REM *** SORRY! GAME OVER ***
1800 POKE54276,33:POKE54277,31
1810 POKE54278,15:FORK=1T04
1820 FORT=50T0120
1830 POKE54273,T:POKE54272,100
1840 NEXT:NEXT
1850 POKE54276,0:SE=0:ESC=0
1860 FORT=1T03:PRINTCHR$(17):NEXT
1870 PRINTTAB(7)CHR$(5)"PRESS A KEY TO PLAY AGAIN"
1880 GETA$:IFA#=""THEN1880
1890 PRINTCHR$(147):GOTO1110
1900 REM *** M/C JOYSTICK DATA ***
1910 DATA120,173,20,3,141,29,192,173,21
1920 DATA3,141,30,192,169,71,141,20,3
1930 DATA169,192,141,21,3,88,96,120,173
1940 DATA29,192,141,20,3,173,30,192,141
1950 DATA21,3,88,96,173,27,192,205,16
1960 DATA192,240,3,76,65,193,169,255
1970 DATA141,27,192,162,0,160,0,185,2
1980 DATA220,141,28,192,169,224,153,2
1990 DATA220,185,0,220,72,173,28,192
2000 DATA153,2,220,189,8,192,221,21,192
2010 DATA189,9,192,253,22,192,144,18
2020 DATA189,8,192,157,21,192,189,9,192
2030 DATA157,22,192,104,74,72,76,162
2040 DATA192,104,74,72,176,13,222,21
2050 DATA192,189,21,192,201,255,208,3
2060 DATA222,22,192,189,21,192,221,12

```

Listing continued next page

```

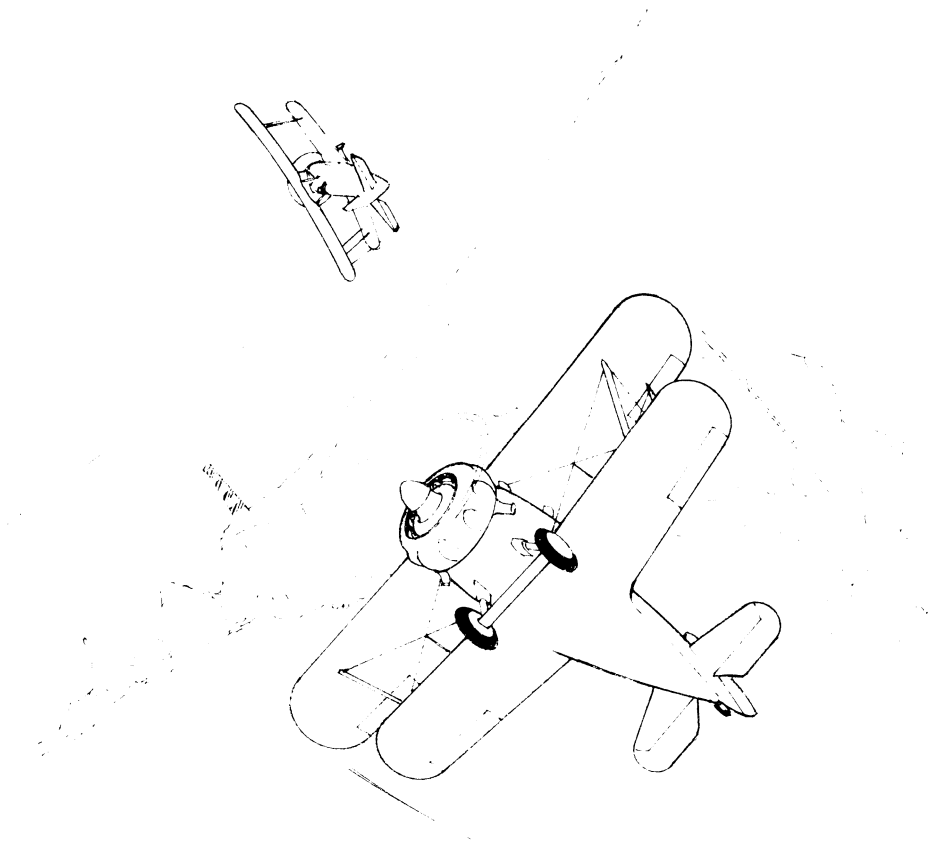
2070 DATA192,189,22,192,253,13,192,144
2080 DATA18,189,12,192,157,21,192,189
2090 DATA13,192,157,22,192,104,74,72,76
2100 DATA207,192,104,74,72,176,8,254,21
2110 DATA192,208,3,254,22,192,189,0,192
2120 DATA221,17,192,189,1,192,253,18
2130 DATA192,144,18,189,0,192,157,17
2140 DATA192,189,1,192,157,18,192,104
2150 DATA74,72,76,1,193,104,74,72,176
2160 DATA13,222,17,192,189,17,192,201
2170 DATA255,208,3,222,18,192,189,17
2180 DATA192,221,4,192,189,18,192,253,5
2190 DATA192,144,18,189,4,192,157,17
2200 DATA192,189,5,192,157,18,192,104
2210 DATA74,72,76,46,193,104,74,72,176
2220 DATA8,254,17,192,208,3,254,18,192
2230 DATA104,74,176,5,169,1,153,25,192
2240 DATA200,232,232,224,4,240,3,76,91
2250 DATA192,238,27,192,108,29,192
2260 DATA35994:REM*CHECKSUM*
2270 REM *** SPRITE 0 (SUB) ***
2280 DATA0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
2290 DATA0,12,0,0,12,0,0,12,0,192,28,0
2300 DATA231,255,240,159,255,248,127
2310 DATA109,188,95,255,248,231,255,240
2320 DATA192,0,0,0,0,0,0,0,0,0,0,0,0,0
2330 DATA0,0,0,0,0,0,0
2340 REM *** SPRITE 1 (SHIP) ***
2350 DATA0,0,0,0,0,0,0,0,0,0,128,0,0
2360 DATA192,0,0,224,0,5,193,32,3,225
2370 DATA200,231,251,242,255,255,255
2380 DATA255,255,255,127,255,254,63,255
2390 DATA254,31,255,252,0,0,0,0,0,0,0,0
2400 DATA0,0,0,0,0,0,0,0,0,0,0,0,0
2410 REM *** SPRITE 2 (BOMB) ***
2420 DATA0,34,0,0,0,0,0,84,0,0,0,1,16,0
2430 DATA0,4,0,0,1,32,0,0,0,0,0,34,0,0

```



```
2440 DATA112,0,0,248,0,0,120,0,0,48,0,0
2450 DATA0,0,0,0,0,0,0,0,0,0,0,0,0,0
2460 DATA0,0,0,0,0,0,0,0
```

Dog Fight



The dog fight takes you back into World War I and puts you in the cockpit of a biplane over the English Channel. Your job is to shoot down as many enemy aircraft as you can. Because of the prevailing wind, you have to be very accurate when targeting the enemy, and you may find you use as many as 30 bullets on one plane. Guide your sight over the plane and shoot like mad!!!

```

1000 REM *** DOG FIGHT ***
1010 REM *** M/C JOY DATA ***
1020 DATA120,173,20,3,141,29,192,173,21
1030 DATA3,141,30,192,169,71,141,20,3
1040 DATA169,192,141,21,3,88,96,120,173
1050 DATA29,192,141,20,3,173,30,192,141
1060 DATA21,3,88,96,173,27,192,205,16
1070 DATA192,240,3,76,65,193,169,255
1080 DATA141,27,192,162,0,160,0,185,2
1090 DATA220,141,28,192,169,224,153,2
1100 DATA220,185,0,220,72,173,28,192
1110 DATA153,2,220,189,8,192,221,21,192
1120 DATA189,9,192,253,22,192,144,18
1130 DATA189,8,192,157,21,192,189,9,192
1140 DATA157,22,192,104,74,72,76,162
1150 DATA192,104,74,72,176,13,222,21
1160 DATA192,189,21,192,201,255,208,3
1170 DATA222,22,192,189,21,192,221,12
1180 DATA192,189,22,192,253,13,192,144
1190 DATA18,189,12,192,157,21,192,189
1200 DATA13,192,157,22,192,104,74,72,76
1210 DATA207,192,104,74,72,176,8,254,21
1220 DATA192,208,3,254,22,192,189,0,192
1230 DATA221,17,192,189,1,192,253,18
1240 DATA192,144,18,189,0,192,157,17
1250 DATA192,189,1,192,157,18,192,104
1260 DATA74,72,76,1,193,104,74,72,176
1270 DATA13,222,17,192,189,17,192,201
1280 DATA255,208,3,222,18,192,189,17
1290 DATA192,221,4,192,189,18,192,253,5
1300 DATA192,144,18,189,4,192,157,17
1310 DATA192,189,5,192,157,18,192,104
1320 DATA74,72,76,46,193,104,74,72,176
1330 DATA8,254,17,192,208,3,254,18,192
1340 DATA104,74,176,5,169,1,153,25,192
1350 DATA200,232,232,224,4,240,3,76,91

```

Listing continued next page

```

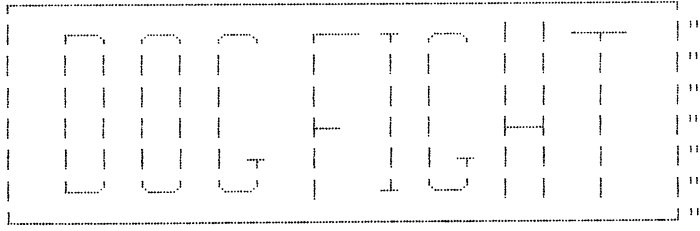
1360 DATA192,238,27,192,108,29,192
1370 REM *** CHECKSUM ***
1380 DATA35994
1390 REM *** READ M/C JOYSTICK ***
1400 FORI=49183TO49478
1410 READJ
1420 POKEI,J
1430 CC=CC+J
1440 NEXTI
1450 READJ
1460 IFCC=JTHEN1480
1470 PRINT"CHECKSUM ERROR":END
1480 SYS(49183):REM INSERT JOY WEDGE
1490 REM ***      (SPRITE 0) ***
1500 FORI=0TO62:READJ
1510 POKE832+I,J:NEXTI
1520 DATA0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
1530 DATA0,0,0,195,0,0,66,0,0,66,0,127
1540 DATA195,254,0,66,0,0,66,0,0,195,0,0
1550 DATA0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
1560 DATA0,0,0,0,0,0,0
1570 REM ***      (SPRITE 1) ***
1580 FORI=0TO62:READJ
1590 POKE896+I,J:NEXTI
1600 DATA0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
1610 DATA24,0,255,255,255,97,24,134,81
1620 DATA90,138,73,60,146,69,126,162,67
1630 DATA126,194,255,255,255,0,60,0,0,90
1640 DATA0,0,129,0,1,129,128,0,0,0,0,0,0
1650 DATA0,0,0,0,0
1660 REM ***      (SPRITE 2) ***
1670 FORI=0TO62:POKEI+960,INT(RND(TI)*255)
1680 NEXTI
1690 REM *** JOYSTICK PARAMETERS ***
1700 Y1=122:X1=155
1710 POKE49152,30:POKE49153,0
1720 POKE49156,250:POKE49157,0

```

```

1730 POKE49160,29:POKE49161,0
1740 POKE49164,229:POKE49165,0
1750 POKE49177,0:POKE49168,0
1760 REM *** VARIABLES & CONSTANTS ***
1770 VC=53248
1780 S=54272
1790 FORI=0TO24:POKES+I,0:NEXTI
1800 POKES+24,15:POKES+5,31
1810 POKES+6,15
1820 REM *** SET UP SPRITES 0 ***
1830 POKE2040,13:POKE2041,14
1840 POKEVC+21,3:REM ENABLE 0
1850 POKEVC+39,1:REM SPRITE 0 COLOR
1860 POKEVC+40,0:REM SPRITE 1 COLOR
1870 POKEVC+29,3:REM EXPAND X SPRITE 0
1880 POKEVC+23,1:REM EXPAND Y SPRITE 1
1890 POKE49169,X1:POKE49173,Y1
1900 POKE49169,X1:POKE49173,Y1
1910 PRINTCHR$(147)
1920 POKE53280,9:POKE53281,9
1930 PRINTCHR$(17)CHR$(17)
1940 PRINT"
1950 PRINT"
1960 PRINT"
1970 PRINT"
1980 PRINT"
1990 PRINT"
2000 PRINT"
2010 PRINT"
2020 PRINTCHR$(17)CHR$(17)CHR$(17)CHR$(5)
2030 PRINTTAB(9)"-PRESS A KEY TO PLAY-"
2040 GETA$:IFA$=""THEN2040
2050 PRINTCHR$(147)
2060 PRINTCHR$(147)
2070 POKE53280,9:POKE53281,14
2080 FORI=0TO400:POKE1624+I,160

```



Listing continued next page

```

2090 POKE55896+I,6:NEXT
2100 X2=50:Y2=50:DX=1:DY=1
2110 POKE2041,14:POKEVC+40,0
2120 POKEVC+2,X2:POKEVC+3,Y2
2130 POKEVC+21,3
2140 REM *** LOOP ***
2150 X1=PEEK(49169):Y1=PEEK(49173)
2160 F=PEEK(49177):POKE49177,0
2170 IFRND(1)<.05THENPOKEVC+23,1:POKEVC+29,1
2180 PRINTCHR$(19)CHR$(5)"SCORE:"SE
2190 IFSE>HITHENHI=SE
2200 PRINTCHR$(19)TAB(25)"HIGH:"HI
2210 IFRND(1)<.05THENPOKEVC+23,3:POKEVC+29,3
2220 IFF<>1THEN2260
2230 POKES+4,129:POKES+1,20:POKES,140
2240 FORK=1TO2:NEXT:POKES+4,0:B=B+1
2250 IFB=500THEN2430
2260 POKEVC+0,X1:POKEVC+1,Y1
2270 POKEVC+2,X2:POKEVC+3,Y2
2280 X2=X2+DX:IFX2>240THENX2=240:DX=-DX
2290 IFX2<50THENX2=50:DX=-DX
2300 Y2=Y2+DY:IFY2>200THENY2=200:DY=-DY
2310 IFY2<50THENY2=50:DY=-DY
2320 IFRND(1)<.1THENDX=DX+1:IFDX>10THENDX=1
2330 IFRND(1)<.1THENDY=DY+1:IFYD>10THENDY=1
2340 IFF=1ANDPEEK(VC+30)AND2=2THENHT=HT+1
2350 IFHT=30THEN2380
2360 GOTO2150
2370 REM *** GOTCHA! ***
2380 POKE2041,15:POKES+4,129:HT=0
2390 POKEVC+40,8:SE=SE+71
2400 POKES+1,7:POKES,90:POKEVC+21,2
2410 FORK=1TO1000:NEXT:POKES+4,0
2420 POKEVC+21,1:F=0:GOTO2100
2430 REM *** SORRY! OUT OF AMMO ***
2440 PRINTCHR$(19)CHR$(17)CHR$(17)CHR$(17)
2450 PRINTTAB(4)"SORRY!. YOU HAVE RUN OUT OF AMMO"

```

```
2460 SE=0:B=0:FORK=1TO2000:NEXT  
2470 POKEVC+21,0  
2480 GOTO1910
```

Spider Hunt



In 'spider hunt' you must destroy as many radioactive spiders as you can, using your laser. Just guide your sight over the spider and press the fire button to watch it turn into an array of colors before it finally dissolves into nothingness. As the sight homes in the spider may start to get itchy feet, so make sure that you fire quickly once you have one at your mercy!!!


```

1000 REM *** SPIDER HUNT ***
1010 PRINTCHR$(147):GOSUB1930:REM TITLE
1020 POKE53280,0:POKE53281,0
1030 REM *** READ JOYSTICK DATA ***
1040 FORI=49183TO49478:READJ
1050 POKEI,J:CC=CC+J:NEXTI
1060 READJ:IFCC<>JTHENPRINT"DATA ERROR":END
1070 SYS(49183):REM INSERT JOYSTICK WEDGE
1080 REM *** READ SPRITE DATA ***
1090 FORI=0TO62:READJ:POKEI+832,J:NEXTI
1100 FORI=0TO62:READJ:POKEI+896,J:NEXTI
1110 REM *** JOYSTICK PARAMETERS ***
1120 POKE49152,40:POKE49153,0
1130 POKE49156,255:POKE49157,0
1140 POKE49160,62:POKE49161,0
1150 POKE49164,213:POKE49165,0
1160 POKE49168,0:POKE49177,0
1170 POKE49169,100:POKE49170,0
1180 POKE49173,150:POKE49174,0
1190 REM *** SET UP SPRITES ***
1200 POKE2040,13:POKE2041,14:VC=53248
1210 POKEVC+23,2:REM EXPAND Y
1220 POKEVC+29,2:REM EXPAND X
1230 POKEVC+2,150:POKEVC+3,150
1240 REM *** SET UP SOUND CHIP ***
1250 S=54272:FORI=0TO24:POKES+L,0:NEXTI
1260 POKES+24,15:REM VOLUME
1270 POKES+5,41:POKES+6,89
1280 POKES+19,31:POKES+20,240
1290 POKE782,7:POKE781,18:SYS65520
1300 PRINTCHR$(30)"* PRESS A KEY TO PLAY *"
1310 GETA$:IFA$=""THEN1310
1320 PRINTCHR$(147)
1330 REM *** SET UP SCREEN ***
1340 PRINTCHR$(19)CHR$(28)CHR$(17)
1350 FORK=1TO22

```

Listing continued next page

[illegible]

```

1730 FORT=128TO1STEP-7:POKEVC+40,T
1740 POKEVC+29,0
1750 POKES+4,33:POKES+0,T:POKES+1,T+127
1760 POKES+18,33:POKES+15,T+7:POKES+14,T+127
1770 POKEVC+29,2
1780 NEXT:POKEVC+21,1
1790 POKES+4,0:POKES+18,0
1800 FORK=1TO1000:NEXT
1810 GOSUB1660
1820 POKE49177,0:POKEVC+40,7
1830 POKEVC+21,3:NP=0
1840 GOTO1510
1850 REM *** SORRY! END GAME ***
1860 PRINTCHR$(19)CHR$(17)CHR$(17)CHR$(17)
1870 PRINTTAB(14)"-GAME OVER-CHR$(17)
1880 PRINTTAB(7)"-PRESS A KEY TO PLAY AGAIN-"
1890 GETA$:IFA$=""THENPOKE646,INT(RND(TI)*15):GOTO1860
1900 PRINTCHR$(147):POKEVC+21,0
1910 NP=0:ED=0:SE=0
1920 PRINTCHR$(147):GOTO1330
1930 REM *** TITLE PAGE ***
1940 PRINTCHR$(30)
1950 PRINTTAB(7)"
1960 PRINTTAB(7)" | 000 00 0 00 000 00 | "
1970 PRINTTAB(7)" | 0 0 0 0 0 0 0 0 | "
1980 PRINTTAB(7)" | 00 00 0 0 0 00 00 | "
1990 PRINTTAB(7)" | 0 0 0 0 0 0 0 0 | "
2000 PRINTTAB(7)" | 000 0 0 00 000 0 0 | "
2010 PRINTTAB(7)"
2020 PRINTTAB(10)"
2030 PRINTTAB(10)" | 0 0 0 0 0 0 000 | "
2040 PRINTTAB(10)" | 0 0 0 0 000 0 0 | "
2050 PRINTTAB(10)" | 000 0 0 0000 0 | "
2060 PRINTTAB(10)" | 0 0 0 0 0 00 0 | "
2070 PRINTTAB(10)" | 0 0 000 0 0 0 | "
2080 PRINTTAB(10)"

```

Listing continued next page

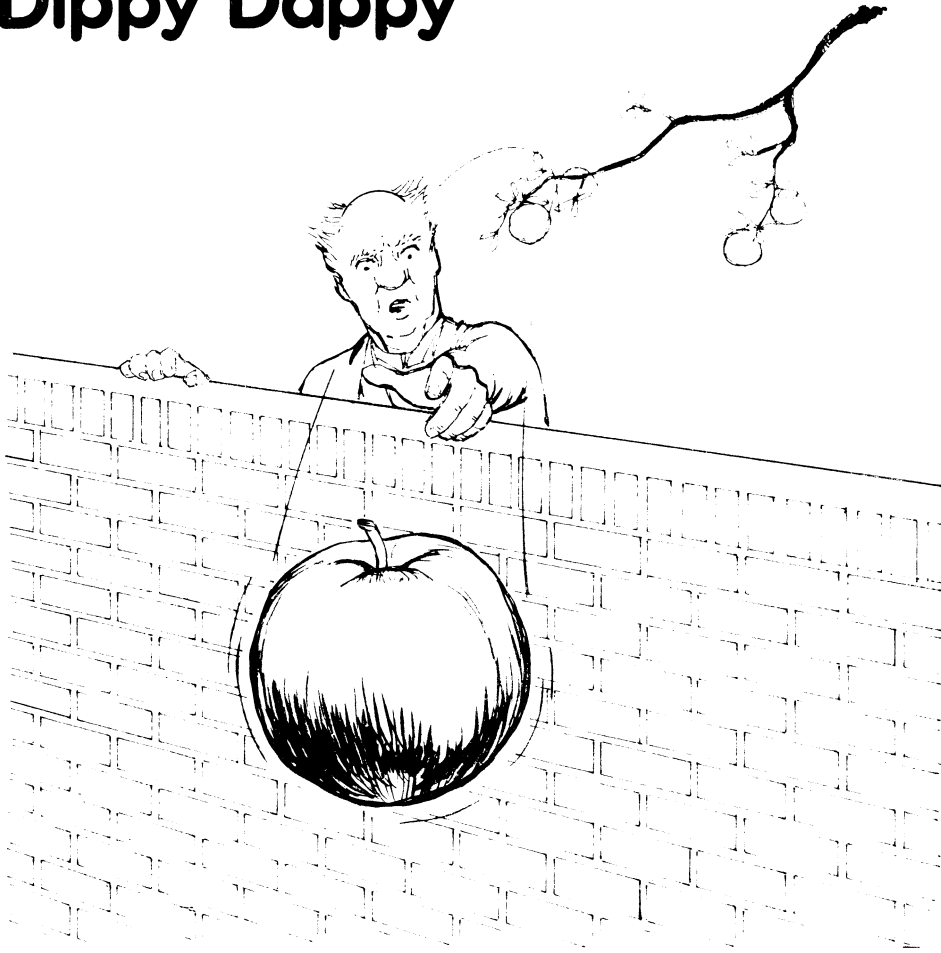
```

2090 RETURN
2100 DATA120,173,20,3,141,29,192,173,21
2110 DATA3,141,30,192,169,71,141,20,3
2120 DATA169,192,141,21,3,88,96,120,173
2130 DATA29,192,141,20,3,173,30,192,141
2140 DATA21,3,88,96,173,27,192,205,16
2150 DATA192,240,3,76,65,193,169,255
2160 DATA141,27,192,162,0,160,0,185,2
2170 DATA220,141,28,192,169,224,153,2
2180 DATA220,185,0,220,72,173,28,192
2190 DATA153,2,220,189,8,192,221,21,192
2200 DATA189,9,192,253,22,192,144,18
2210 DATA189,8,192,157,21,192,189,9,192
2220 DATA157,22,192,104,74,72,76,162
2230 DATA192,104,74,72,176,13,222,21
2240 DATA192,189,21,192,201,255,208,3
2250 DATA222,22,192,189,21,192,221,12
2260 DATA192,189,22,192,253,13,192,144
2270 DATA18,189,12,192,137,21,192,189
2280 DATA13,192,157,22,192,104,74,72,76
2290 DATA207,192,104,74,72,176,8,254,21
2300 DATA192,208,3,254,22,192,189,0,192
2310 DATA221,17,192,189,1,192,253,18
2320 DATA192,144,18,189,0,192,157,17
2330 DATA192,189,1,192,157,18,192,104
2340 DATA74,72,76,1,193,104,74,72,176
2350 DATA13,222,17,192,189,17,192,201
2360 DATA255,208,3,222,18,192,189,17
2370 DATA192,221,4,192,189,18,192,253,5
2380 DATA192,144,18,189,4,192,157,17
2390 DATA192,189,5,192,157,18,192,104
2400 DATA74,72,76,46,193,104,74,72,176
2410 DATA8,254,17,192,208,3,254,18,192
2420 DATA104,74,176,5,169,1,153,25,192
2430 DATA200,232,232,224,4,240,3,76,91
2440 DATA192,238,27,192,108,29,192
2450 DATA35994:REM*CHECKSUM*
2460 REM *** SPRITE 0 (SIGHT) ***
2470 DATA0,0,0,31,1,240,24,0,48,20,0,80
2480 DATA18,0,144,17,41,16,0,170,0,0
2490 DATA108,0,1,239,0,0,0,0,1,239,0,0
2500 DATA108,0,0,170,0,17,41,16,18,0
2510 DATA144,20,0,80,24,0,48,31,1,240
2520 DATA0,0,0,0,0,0,0,0,0,0

```

```
2530 REM *** SPRITE 1 (THING) ***
2540 DATA0,0,0,0,0,0,0,0,0,24,97,128
2550 DATA102,246,96,129,216,16,15,127
2560 DATA0,51,236,192,199,190,48,30,247
2570 DATA128,99,252,96,130,100,16,15
2580 DATA111,0,48,240,192,65,104,32,130
2590 DATA4,16,128,0,16,0,0,0,0,0,0,0
2600 DATA0,0,0,0
```

Dippy Dappy



'O.K. you chuck the apples over Dappy, and I'll catch them!'

'Derrr, right boss ...'

Well he did say chuck them over didn't he? And he did tell me where to throw them, didn't he?

The story so far: Dappy, at the request of his boss Midget Malone, has broken into Farmer Giles's orchard, intent on stealing his nice rosy apples. Dappy, however, has never had all that much schooling and was never told exactly what an apple looked like. So you can understand it was not his fault when he started throwing eggs over!

Where did he get the eggs from? I hear you ask ... well this is only a fairy story, after all!

With the aid of a joystick, try and manoeuvre Malone underneath falling eggs and catch as many as you can. Drop too many and the game ends!

Have a smashing time playing 'Dippy Dappy'.

```

1000 PRINTCHR$(147)
1010 REM *** READ M/C JOYSTICK DATA ***
1020 FORI=49183TO49478:READJ
1030 POKEI,J:CC=CC+J:NEXTI
1040 READJ:IFJ<>CCTHENPRINT"DATA ERROR":END
1050 REM *** READ SPRITE DATA 0,1,2 ***
1060 FORI=0TO62:READJ:POKEI+832,J:NEXT
1070 FORI=0TO62:READJ:POKEI+896,J:NEXT
1080 FORI=0TO62:READJ:POKEI+960,J:NEXT
1090 REM *** JOYSTICK PARAMETERS ***
1100 POKE49152,30:POKE49153,0
1110 POKE49156,255:POKE49157,0
1120 POKE49160,180:POKE49161,0
1130 POKE49164,197:POKE49165,0
1140 POKE49168,0:POKE49169,255:POKE49173,180
1150 REM *** SET UP SPRITES 0,1,2 ***
1160 POKE2040,13:POKE2041,14:POKE2042,15
1170 VC=53248:SYS49183
1180 POKEVC+0,120:POKEVC+1,68:REM POS.0
1190 POKEVC+2,120:POKEVC+3,160:REM POS.1
1200 POKEVC+29,3:REM EXPANDX 0,1
1210 POKEVC+23,1:REM EXPANDY 0,1
1220 POKEVC+39,5:REM COLOR SPRITE 0
1230 POKEVC+40,3:REM COLOR SPRITE 1
1240 POKEVC+41,1:REM COLOR SPRITE 2
1250 X=120
1260 REM *** SET UP SCREEN ***
1270 POKE53280,0:POKE53281,0
1280 REM POKE53265,PEEK(53265)AND239
1290 SC=1064:CO=55336
1300 FORT=0TO160:POKECO+T,6
1310 POKESC+T,160:NEXTT
1320 FORT=0TO559STEP3:POKECO+T+160,2
1330 POKECO+T+160+1,2
1340 POKECO+T+160+2,2:POKESC+T+160,239
1350 POKESC+T+160+1,239

```

Listing continued next page

```

1360 POKE SC+T+160+2,250:NEXT T
1370 FORT=0T0240:POKE CO+T+720,5
1380 POKE SC+T+720,160:NEXT
1390 PRINT CHR$(19)CHR$(30)
1400 PRINT TAB(30)CHR$(18)"  "CHR$(149);
1410 PRINT"  "CHR$(28)"  "CHR$(30)"  ";
1420 PRINT CHR$(149)"  "CHR$(30)"  "
1430 PRINT CHR$(18);
1440 PRINT TAB(29)"  "CHR$(28)"  "CHR$(30);
1450 PRINT"  "CHR$(28)"  "CHR$(30);
1460 PRINT CHR$(18);
1470 PRINT"  "CHR$(149)"  "CHR$(30)"  "
1480 PRINT CHR$(18);
1490 PRINT TAB(29)CHR$(28)"  "CHR$(30)"  ";
1500 PRINT CHR$(149)"  "CHR$(30)"  "CHR$(149);
1510 PRINT"  "CHR$(28)"  "CHR$(30)"  ";
1520 PRINT CHR$(149)"  "CHR$(28)"  "CHR$(30);
1530 PRINT"  "
1540 PRINT CHR$(18);
1550 PRINT TAB(28)CHR$(28)"  "CHR$(30)"  ";
1560 PRINT CHR$(28)"  "CHR$(149)"  "CHR$(30);
1570 PRINT"  "CHR$(149)"  "CHR$(30)"  ";
1580 PRINT CHR$(149)"  "
1590 REM POKE53265,PEEK(53265)OR16
1600 POKE VC+21,3:REM ENABLE SPRITES 0,1,2
1610 POKE782,32:POKE781,10:SYS65520
1620 PRINT CHR$(5)CHR$(18)"  _____"
1630 POKE782,32:POKE781,11:SYS65520
1640 PRINT CHR$(5)CHR$(18)" |KEEP |"
1650 POKE782,32:POKE781,12:SYS65520
1660 PRINT CHR$(5)CHR$(18)" |OUT! |"
1670 POKE782,32:POKE781,13:SYS65520
1680 PRINT CHR$(5)CHR$(18)"  _____"CHR$(146)
1690 PRINT CHR$(19)CHR$(158)"SCORE:"
1700 PRINT CHR$(19)CHR$(158)TAB(17)"HIGH SCORE:"
1710 D=8:GOSUB1930:GOSUB1970
1720 REM *** MAIN LOOP ***

```



```

1730 PRINTCHR$(19)CHR$(5)TAB(8);SE;" "
1740 IFSE>HITHENHI=SE
1750 PRINTCHR$(19)CHR$(5)TAB(30);HI
1760 POKEVC+0,X
1770 IFDR=0THENAX=INT(RND(TI)*210+40)
1780 IFDR=0THENF=F-1:IFF<1THENF=1
1790 X=X+D:IFX<300ORX>246THEND=-D
1800 POKEVC+39,INT(RND(TI)*2+4)
1810 POKEVC+30,0
1820 PX=PEEK(49169):PY=PEEK(49173)
1830 IFPEEK(VC+30)AND2=2THENGOSUB1970
1840 IFF=1THENDR=1
1850 IFDR=1THENPOKEVC+21,7
1860 IFDR=1THENAY=AY+6+INT(SE/100)
1870 IFAY>190THENGOSUB1930
1880 POKEVC+2,PX:POKEVC+3,PY
1890 POKEVC+4,AX:POKEVC+5,AY
1900 IFCA=1THENSE=SE+10:CA=0
1910 GOTO1730
1920 REM *** DROP ANOTHER! ***
1930 AY=95:POKEVC+21,3:DR=0
1940 IFCA=0THENMI=MI+1:IFMI>5THEN2000
1950 RETURN
1960 REM *** CAUGHT! ***
1970 CA=1:AY=95:DR=0
1980 POKEVC+21,3:F=2
1990 RETURN
2000 REM *** SORRY! DROPPED TOO MANY ***
2010 POKE781,20:POKE782,6:SYS65520
2020 PRINTCHR$(5)"SORRY DROPPED ONE TOO MANY"
2030 POKE781,22:POKE782,6:SYS65520
2040 PRINTCHR$(5)"PRESS A KEY TO PLAY AGAIN"
2050 GETA$:IFA#=""THEN2050
2060 SE=0:DR=0:F=5:AY=85:CA=0:MI=0
2070 GOTO1370
2080 DATA120,173,20,3,141,29,192,173,21

```

Listing continued next page

```

2090 DATA3,141,30,192,169,71,141,20,3
2100 DATA169,192,141,21,3,88,96,120,173
2110 DATA29,192,141,20,3,173,30,192,141
2120 DATA21,3,88,96,173,27,192,205,16
2130 DATA192,240,3,76,65,193,169,255
2140 DATA141,27,192,162,0,160,0,185,2
2150 DATA220,141,28,192,169,224,153,2
2160 DATA220,185,0,220,72,173,28,192
2170 DATA153,2,220,189,8,192,221,21,192
2180 DATA189,9,192,253,22,192,144,18
2190 DATA189,8,192,157,21,192,189,9,192
2200 DATA157,22,192,104,74,72,76,162
2210 DATA192,104,74,72,176,13,222,21
2220 DATA192,189,21,192,201,255,208,3
2230 DATA222,22,192,189,21,192,221,12
2240 DATA192,189,22,192,253,13,192,144
2250 DATA18,189,12,192,157,21,192,189
2260 DATA13,192,157,22,192,104,74,72,76
2270 DATA297,192,104,74,72,176,8,254,21
2280 DATA192,208,3,254,22,192,189,0,192
2290 DATA221,17,192,189,1,192,253,18
2300 DATA192,144,18,189,0,192,157,17
2310 DATA192,189,1,192,157,18,192,104
2320 DATA74,72,76,1,193,104,74,72,176
2330 DATA13,222,17,192,189,17,192,201
2340 DATA255,208,3,222,18,192,189,17
2350 DATA192,221,4,192,189,18,192,253,5
2360 DATA192,144,18,189,4,192,157,17
2370 DATA192,189,5,192,157,18,192,104
2380 DATA74,72,76,46,193,104,74,72,176
2390 DATA8,254,17,192,208,3,254,18,192
2400 DATA104,74,176,5,169,1,153,25,192
2410 DATA200,232,232,224,4,240,3,76,91
2420 DATA192,238,27,192,106,29,192
2430 DATA35994:REM*CHECKSUM*
2440 REM *** SPRITE 0 (DAPPY) ***
2450 DATA12,156,152,98,127,35,17,255

```

```

2460 DATA196,11,255,232,103,255,243,31
2470 DATA127,124,14,62,56,30,62,60,28
2480 DATA255,156,28,190,156,31,255,252
2490 DATA0,62,0,0,127,0,0,99,0,0,193
2500 DATA128,0,128,128,0,65,0,0,62,0
2510 DATA0,0,0,0,0,0,0,0,0,0
2520 REM *** SPRITE 1 (CATCH) ***
2530 DATA0,0,0,15,255,224,18,170,144,19
2540 DATA255,144,13,85,96,15,255,224,24
2550 DATA254,48,48,0,24,96,254,12,99
2560 DATA255,140,103,17,204,62,68,248
2570 DATA30,0,240,3,57,128,3,255,128,3
2580 DATA199,128,1,255,0,0,108,0,0,108
2590 DATA0,1,239,0,3,239,128
2600 REM *** SPRITE 2 (BOMB) ***
2610 DATA0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
2620 DATA0,0,0,0,32,0,0,32,0,0,32,0,0
2630 DATA32,0,1,36,0,1,36,0,1,4,0,1,56
2640 DATA128,1,124,128,0,254,128,0,254
2650 DATA0,0,124,0,0,56,0,0,0,0,0,0,0

```

Fun Match



Fun Match is another short program in which you have a set of three words that can be mixed up to form other words. Sometimes you get quite funny results.

The DATA statements can be changed to words of your own choice. The words we have used are just an example and you are not limited to ten sets of words. This program can be used as a basis for a much larger program, i.e. Anagrams or Hangman.

```

1000 REM *** SET UP ARRAYS ***
1010 DIM F$(10), S$(10), T$(10)
1020 FOR I=1 TO 10
1030 READ F$(I), S$(I), T$(I)
1040 NEXT I
1050 REM *** MAIN PROGRAM LOOP ***
1060 PRINT CHR$(147)
1070 N1=INT(RND(TI)*10+1)
1080 N2=INT(RND(TI)*10+1)
1090 N3=INT(RND(TI)*10+1)
1100 PRINT "THE "; F$(N1);
1110 PRINT " "; S$(N2); " "; T$(N3)
1120 PRINT:PRINT:PRINT
1130 PRINT "PRESS A KEY"
1140 GET A$: IF A$="" THEN 1140
1150 GOTO 1060
1160 REM *** DATA FOR WORDS ***
1170 DATA "BIG", "BROWN", "DOG"
1180 DATA "SMALL", "ROUND", "BALL"
1190 DATA "FUNNY", "SMILING", "CLOWN"
1200 DATA "JOLLY", "FAT", "MAN"
1210 DATA "DARK", "BLUE", "SEA"
1220 DATA "SHRIVELLED", "GREEN", "BEAN"
1230 DATA "LONELY", "ANT", "EATER"
1240 DATA "BRIGHT", "YOUNG", "EXECUTIVE"
1250 DATA "FAST", "SALOON", "CAR"
1260 DATA "SOGGY", "PINK", "SOCK"

```

Guess my number



Starting off with a simple program is a good idea, and with 'Guess my number' not only can you play it, but you can also learn about the use of variables and IF ... THEN statements.

The idea of the game is that the computer picks a random number, and you have to guess it. The range of the number can be anywhere between 0 and 100. Of course, you can change this to your own requirements if you wish. Each time you give a guess the Computer responds with either 'Too high' or 'Too low', in which case your next guess will change accordingly.

When you have finally guessed the correct number, the Computer will tell you how many guesses you have taken.

The actual listing is barely a screen full, and proves that you don't need to use a lot of memory to do something worthwhile.

```

1000 REM *** GUESS MY NUMBER ***
1010 PRINTCHR$(147);GG=0;Poke5280,6
1020 PRINTCHR$(5)"GUESS MY NUMBER BETWEEN 1 & 100"
1030 X=INT(RND(1)*100)+1
1040 Poke19,1 NM1?
1050 PRINTPRINTCHR$(154)"TYPE IN GUESS > ";
1060 IFGG=5:GG=INT(VAL(G$));
1070 IFGG<XTHENPRINTPRINTPRINTCHR$(155)"GUESS TOO
LOW TRY AGAIN":GG=GG+1
1080 IFGG>XTHENPRINTPRINTPRINTCHR$(155)"GUESS TOO
HIGH TRY AGAIN":GG=GG+1
1090 IFGG=XTHENPRINTPRINTPRINTCHR$(155)"GOOD, YOU GUESSED
IN "GG" GOES"
1100 IFGG=XTHENPRINTPRINT"ANOTHER GO? (Y/N)"
1110 IFGG<XTHEN1040
1120 IFGG=X:IFGG<"Y"ANDGG<"N"THEN100
1130 IFGG="Y"THEN1040
1140 PRINTCHR$(147)CHR$(154)
1150 Poke5280,14END

```

Bug Bovver



For those amongst us who suffer from terminal termite trouble or are driven insane by insects, this game lets them get their own back.

When the program is RUN the boot weaves menacingly back and forth across the screen. To drop the boot press the SPACE bar. Listen to the soggy splattering sound with just a hint of crunch as the bug gets its come-uppance!


```

1000 REM *** BUG BOVVER ***
1010 PRINTCHR$(147)
1020 POKE53280,0:POKE53281,0
1030 GOSUB1720
1040 PRINTCHR$(152)
1050 TI$="0000000"
1060 FORI=0TO62:READJ:POKEI+832,J:NEXT
1070 FORI=0TO62:READJ:POKEI+896,J:NEXT
1080 FORI=0TO62:READJ:POKEI+960,J:NEXT
1090 POKE2040,13:POKE2041,14
1100 VC=53248
1110 POKEVC+23,6:POKEVC+29,6
1120 POKEVC+39,9
1130 POKEVC+40,5
1140 POKEVC+41,5
1150 POKEVC+37,2:POKEVC+38,7
1160 BD=8:DY=8:BY=60:POKEVC+1,BY
1170 GD=4:POKEVC+3,200
1180 BX=INT(RND(1)*140+56)
1190 GX=INT(RND(1)*115+66)
1200 REM***BUG BOVVER***
1210 S=54272:D=-1:SO=38
1220 POKES+24,14
1230 POKES+5,31:POKES+6,240
1240 POKE781,22:POKE782,0:SYS65520
1250 FORI=1TO40:PRINT"■":NEXT
1260 PRINTCHR$(19)"SCORE:"SPC(21)"HIGH:"
1270 PRINTCHR$(19):FORI=1TO120
1280 PRINTCHR$(18)" ";NEXT
1290 POKEVC+21,3
1300 REM *** MAIN LOOP ***
1310 BX=BX+BD:IFBX<550RBX>246THENBD=-BD
1320 POKES+4,0:POKEVC+30,0
1330 POKEVC,BX:POKEVC+1,BY
1340 GX=GX+GD:IFGX<450RGX>240THENGD=-GD
1350 POKEVC+2,GX

```

Listing continued next page


```

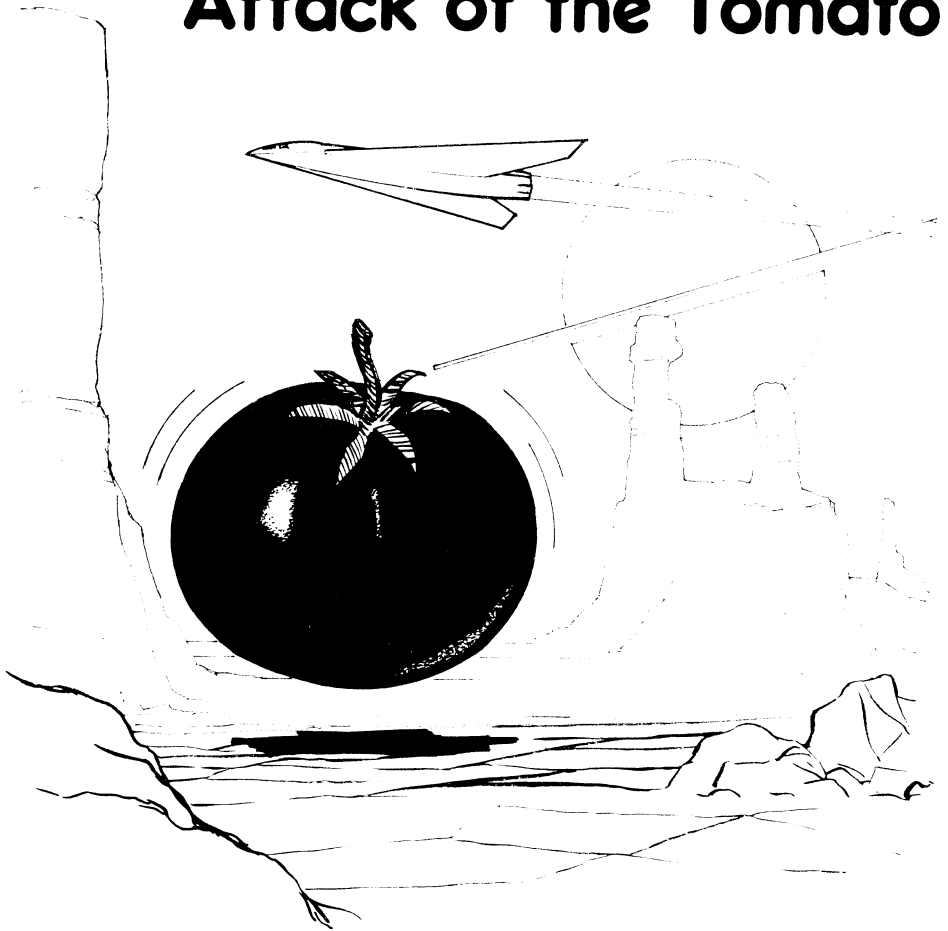
1730 PRINTCHR$(17)CHR$(17)CHR$(30)
1740 PRINTTAB(8)"  O  BOOOOOO  "
1750 PRINTTAB(8)"HIII  HIIIIIIII  "
1760 PRINTTAB(8)"OOO  OOOIIIIK  "
1770 PRINTCHR$(17)CHR$(17)CHR$(17)
1780 PRINTTAB(9)"PRESS A KEY TO PLAY"
1790 GETB$;IFB$=""THEN1790
1800 PRINTCHR$(147):RETURN
1810 REM *** SORRY! OUT OF TIME ***
1820 POKES+4.0
1830 PRINTCHR$(19)CHR$(17)CHR$(17)CHR$(17)
1840 PRINTCHR$(17)
1850 PRINTTAB(9)"--SORRY! OUT OF TIME--"
1860 PRINTCHR$(17)CHR$(17)CHR$(17)
1870 PRINTTAB(7)"PRESS A KEY TO PLAY AGAIN"
1880 GETB$;IFB$=""THEN1880
1890 TI$="0000000"
1900 SC=0:PRINTCHR$(147):GOTO1090
1910 REM *** SPRITE 0 (BOOT) ***
1920 DATA0,0,0,124,0,0,63,248,0,63,248
1930 DATA0,63,248,0,31,192,0,31,240,0
1940 DATA31,192,0,31,240,0,15,192,0,15
1950 DATA248,0,15,228,0,31,254,0,31,249
1960 DATA176,63,255,184,0,15,188,127
1970 DATA240,188,127,255,0,127,143,254
1980 DATA127,128,254,0,0,0
1990 REM *** SPRITE 1 (BUG) ***
2000 DATA0,0,0,0,124,0,0,254,0,1,187,0
2010 DATA3,17,128,3,1,128,7,69,192,7,17
2020 DATA192,7,255,192,1,199,0,0,124,0
2030 DATA0,108,0,0,108,0,0,108,0,3,239
2040 DATA128,0,0,0,0,0,0,0,0,0,0,0,0
2050 DATA0,0,0,0,0
2060 REM *** SPRITE 2 (SPLAT) ***
2070 DATA0,0,0,0,0,0,0,0,0,0,0,0,0,0
2080 DATA0,0,0,0,0,0,0,0,0,0,0,0,0,0

```

Listing continued next page

2090 DATA0,0,0,7,255,224,62,90,124,127
2100 DATA255,254,0,195,0,7,195,224,0,0
2110 DATA0,0,0,0,0,0,0,0,0,0,0,0

Attack of the Tomato



Your city is under attack from fertilizer-crazed tomatoes. These vituperative vegetables are deadly!
As commander of a front-line outpost your mission:

TURN THOSE TOMATOES TO RATATOUILLE!!

Let more than five past and your city becomes KETCHUP.


```

1220 GETA$:IF A$=""THEN1220
1230 V=53248:N=4:LX2=100:MX=0:Y2=100
1240 GOSUB1940:REM B/L FOR JOYSTICK
1250 PRINTCHR$(147):PRINTCHR$(5):REM CLEAR SCREEN
1260 GOSUB1780:REM SET UP SCREEN
1270 REM----SPRITES----
1280 REM--SET MEM POINTERS--
1290 POKE2040,192:POKE2041,193
1300 POKE2042,195:POKE2044,196
1310 REM--READ SPRITE DATA--
1320 FORI=12288TO12350:READA:POKEI,A:NEXT
1330 FORI=12352TO12414:READA:POKEI,A:NEXT
1340 FORI=12416TO12478:READA:POKEI,A:NEXT
1350 FORI=12480TO12542:READA:POKEI,A:NEXT
1360 FORI=12544TO12606:READA:POKEI,A:NEXT
1370 FORI=12736TO12798:READA:POKEI,A:NEXT
1380 REM--COLOR SPRITES--
1390 POKEV+41,1:POKEV+43,7:POKEV+39,5
1400 REM--EXPAND SPRITES--
1410 POKEV+23,3:POKEV+29,11
1420 REM--TURN ON SPRITES--
1430 POKEV+21,7
1440 REM--SET JOYSTICK LIMITS--
1450 POKE49152,0:POKE49153,0:REM X2 MIN
1460 POKE49156,65:POKE49157,1:REM X2 MAX
1470 POKE49160,47:POKE49161,0:REM Y2 MIN
1480 POKE49164,232:POKE49165,0:REM Y2 MIN
1490 POKE49168,0:REM SCAN RATE
1500 SYS49183:REM TURN ON WEDGE CODE
1510 REM----MAIN MOVEMENT ROUTINE-----
1520 REM----RESET TOMATO----
1530 X0=345:POKEV+16,PEEK(V+16)OR3
1540 POKE49169,LX2:POKE49170,MX
1550 POKE49173,Y2:POKE49174,0
1560 POKEV,90:POKEV+2,90
1570 POKE2041,193:POKEV+40,2:REM RESET POINTER AND
    COLOR

```

Listing continued next page

```

1580 POKEV+21,PEEK(V+21)OR3:REM TURN TOM BACK ON
1590 REM-----MOVE LOOP-----
1600 X0=X0-N:Y0=140-40*SIN(X0/N)
1610 PRINTTAB(3)CHR$(145)"LET THRU";T,"YOUR SCORE"
;YS
1620 GOSUB2690:REM SHIP CONTROL
1630 Y1=Y0+14
1640 HX=INT(X0/256):LX=X0-256*HX
1650 IF HX=0THEN POKEV+16,PEEK(V+16)AND252
1660 POKEV,LX:POKEV+1,Y0:REM SPR 0 POS
1670 POKEV+2,LX:POKEV+3,Y1:REM SPR 1 POS
1680 GOSUB2690:REM SHIP CONTROL
1690 IFC>=5THENC=0:YS=YS+100:N=N+2:GOTO1530
1700 REM---IF TOMATO ESCAPES---
1710 IFX0<10THENT=T+1:C=0:GOTO1530
1720 IFT>5THEN1740:REM NO. THRU>5
1730 GOTO1600
1740 PRINTCHR$(147):GOSUB2880:REM END OF GAME
1750 SYS49208:REM DISABLE WEDGE
1760 RUN1230
1770 END
1780 REM-----SET UP SCREEN-----
1790 POKE53280,0:REM COLOR BORDER
1800 POKE53281,6:REM COLOR SCREEN
1810 REM---READ BACKGROUND DATA---
1820 FORI=1464TO1663:READA:POKEI,A
1830 POKE54272+I,4:NEXT
1840 FORI=1664TO1743:POKEI,160
1850 POKE54272+I,4:NEXT
1860 FORI=1744TO2023:POKEI,160
1870 POKE54272+I,14:NEXT
1880 REM---STARS---
1890 FORI=1TO40
1900 P=1024+INT(RND(1)*440)
1910 POKEP,46:POKEP+54272,1
1920 NEXT
1930 RETURN

```



```

1940 REM-----B/L JOYSTICK-----
1950 DATA120,173,20,3,141,29,192,173,21
1960 DATA3,141,30,192,169,71,141,20,3
1970 DATA169,192,141,21,3,88,96,120,173
1980 DATA29,192,141,20,3,173,30,192,141
1990 DATA21,3,88,96,173,27,192,205,16
2000 DATA192,240,3,76,65,193,169,255
2010 DATA141,27,192,162,0,160,0,185,2
2020 DATA220,141,28,192,169,224,153,2
2030 DATA220,185,0,220,72,173,28,192
2040 DATA153,2,220,189,8,192,221,21,192
2050 DATA189,9,192,253,22,192,144,18
2060 DATA189,8,192,157,21,192,189,9,192
2070 DATA157,22,192,104,74,72,76,162
2080 DATA192,104,74,72,176,13,222,21
2090 DATA192,189,21,192,201,255,208,3
2100 DATA222,22,192,189,21,192,221,12
2110 DATA192,189,22,192,253,13,192,144
2120 DATA18,189,12,192,157,21,192,189
2130 DATA13,192,157,22,192,104,74,72,76
2140 DATA207,192,104,74,72,176,8,254,21
2150 DATA192,208,3,254,22,192,189,0,192
2160 DATA221,17,192,189,1,192,253,18
2170 DATA192,144,18,189,0,192,157,17
2180 DATA192,189,1,192,157,18,192,104
2190 DATA74,72,76,1,193,104,74,72,176
2200 DATA13,222,17,192,189,17,192,201
2210 DATA255,208,3,222,18,192,189,17
2220 DATA192,221,4,192,189,18,192,253,5
2230 DATA192,144,18,189,4,192,157,17
2240 DATA192,189,5,192,157,18,192,104
2250 DATA74,72,76,46,193,104,74,72,176
2260 DATA8,254,17,192,208,3,254,18,192
2270 DATA104,74,176,5,169,1,153,25,192
2280 DATA200,232,232,224,4,240,3,76,91
2290 DATA192,238,27,192,108,29,192

```

Listing continued next page

```

2300 DATA35994:REM*CHECKSUM*
2310 FORI=49183TO49478
2320 READX:POKEI,X:CC=CC+X
2330 NEXT
2340 READX:IFX<>CCTHEN PRINT"CHECKSUM ERROR."
2350 RETURN
2360 REM-----FIRE SUBROUTINE-----
2370 X4=X2+12
2380 POKEV+21,PEEK(V+21)OR16:REM TURN ON
2390 YS=YS-10
2400 IFPEEK(2042)=194THEN DX=-30
2410 L=0
2420 IFPEEK(2042)=195THEN DX=30
2430 X4=X4+DX
2440 L=L+1:IFL>4THENGOTO2540
2450 H4=INT(X4/256):L4=X4-256*H4
2460 IFH4=1THENPOKEV+16,PEEK(V+16)OR16
2470 IFH4=0THENPOKEV+16,PEEK(V+16)AND239
2480 POKEV+8,L4
2490 REM SCREEN LIMITS
2500 IFX4<0ORX4>320THENPOKEV+21,PEEK(V+21)AND239:G
OTO2540
2510 PC=PEEK(V+30):REM COLLISION REG
2520 IFPC=19THENPOKEV+21,PEEK(V+21)AND239:GOSUB255
0
2530 GOTO2430
2540 RETURN
2550 REM-----COLLISION ROUTINE-----
2560 C=C+0.5
2570 REM---CHANGE TOM COLOUR---
2580 ONC+1GOTO2620,2630,2640,2650,2660
2590 REM---EXPLODE TOMATO AND TURN OFF---
2600 POKE2041,199:FORI=1TO2000:NEXT
2610 POKEV+21,PEEK(V+21)AND252:GOTO2670
2620 POKEV+40,8:GOTO2670
2630 POKEV+40,10:GOTO2670
2640 POKEV+40,9:GOTO2670

```



```

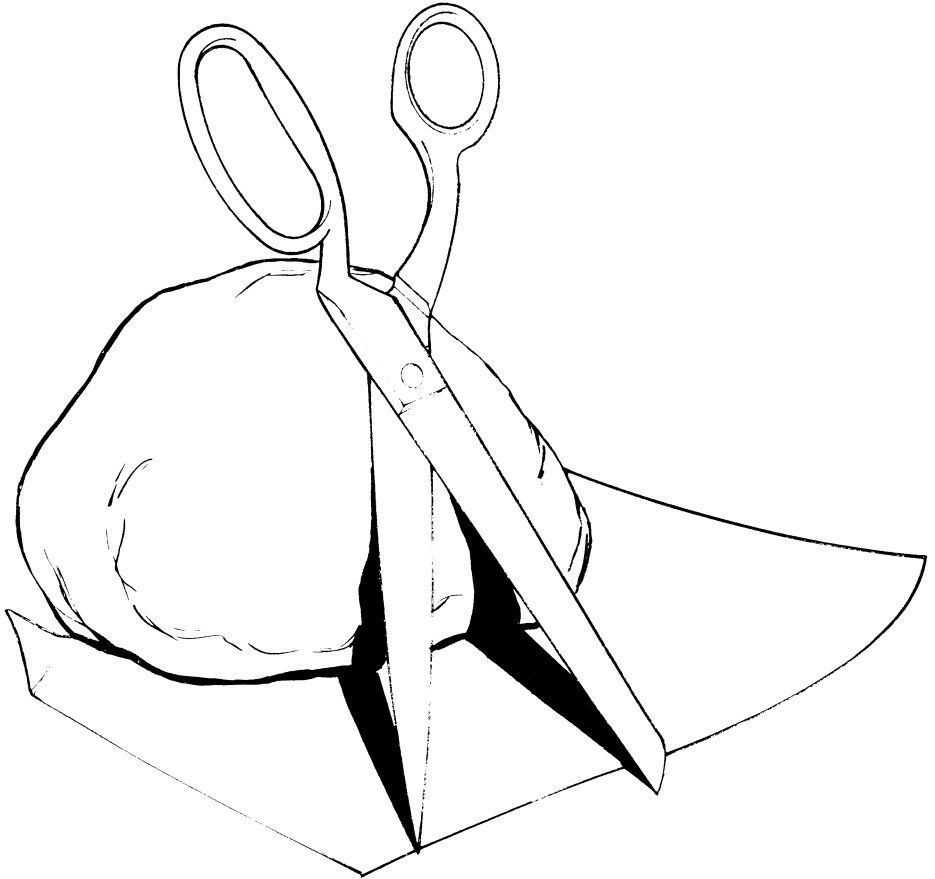
3310 DATA0,0,0,0,0,0,0,0,0,0
3320 DATA192,0,0,192,0,0,192,0,0,224
3330 DATA0,0,96,0,60,115,192,39,54,32
3340 DATA1,252,0,3,159,0,6,3,128,4,0
3350 DATA128,4,0,128
3360 DATA0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
3370 DATA0,0,0
3380 REM-----SPRITE 1 DATA-----
3390 DATA0,0,0,15,131,192,31,199,224
3400 DATA63,199,240,63,239,248,127,255
3410 DATA252,127,255,252,255,255,254
3420 DATA255,255,254,255,255,254,255
3430 DATA255,254,255,255,254,255,255
3440 DATA252,127,255,252,127,255,248
3450 DATA63,255,248,63,255,240,31,255
3460 DATA224,15,255,192,7,255,0,1,252,0
3470 REM-----SPRITE 2 DATA-----
3480 DATA0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
3490 DATA7,0,0,63,0,0,237,0,7,152,0,28
3500 DATA48,0,125,224,7,255,252,3,255
3510 DATA252,0,125,224,0,28,48,0
3520 DATA7,152,0,0,237,0,0,63,0,0,7
3530 DATA0,0,0,0,0,0,0,0,0,0
3540 REM-----SPRITE 2B DATA-----
3550 DATA0,0,0,0,0,0,0,0,0,0
3560 DATA224,0,0,252,0,0,183,0,0,25,224
3570 DATA0,12,56,0,7,190,0,63,255,224
3580 DATA63,255,192,7,190,0,12,56,0,25
3590 DATA224,0,183,0,0,252,0,0,224
3600 DATA0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
3610 REM-----SPRITE 4 DATA-----
3620 DATA0,0,0,0,0,0,0,0,0,0
3630 DATA0,0,0,0,0,0,0,0,0,0
3640 DATA0,0,0,0,0,0,0,0,0,0
3650 DATA195,12,48,195,12,48
3660 DATA0,0,0,0,0,0,0,0

```

Listing continued next page

```
3670 DATA0,0,0,0,0,0,0,0
3680 DATA0,0,0,0,0,0,0,0
3690 DATA0,0,0,0,0,0,0,0
3700 REM----SPRITE 1B DATA----
3710 DATA1,16,8,64,0,16,32,0
3720 DATA0,16,16,64,8,0,128,0,65
3730 DATA0,66,16,4,1,16,16,4,128
3740 DATA64,0,19,0,73,56,0,0
3750 DATA108,105,4,56,0,128,16,0
3760 DATA0,130,16,1,17,1,64,0,128
3770 DATA0,4,0,8,0,32,16,82,16,32,16,0
```

Paper, Rock, Scissors



Beat the computer at this old children's playground game. The rules are simple:

Scissors cut paper
Rock blunts scissors
Paper wraps rock

When you choose either paper, rock, or scissors – by pressing respectively P, R or S – the computer also makes a random choice. If, for example, you choose scissors and the computer chooses paper then you win. If, however, the computer chooses rock then it wins – because rock blunts scissors.

```

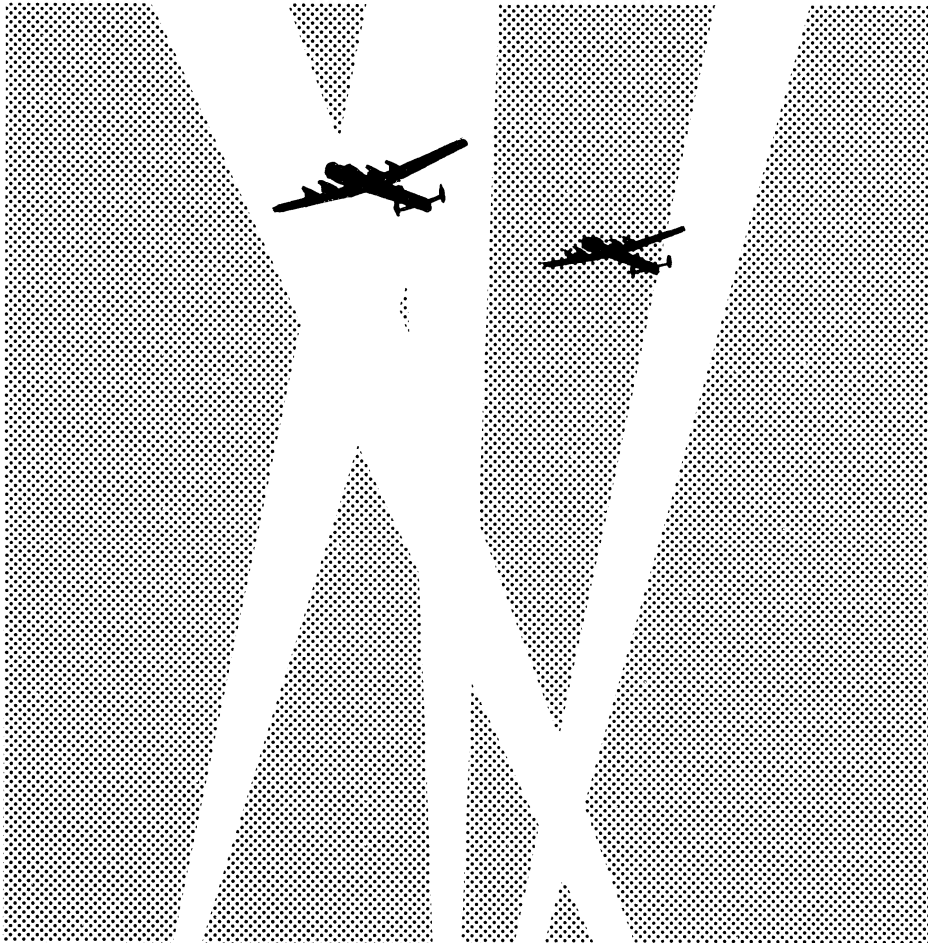
1000 REM *** PAPER ROCK SCISSORS ***
1010 PRINTCHR$(147)
1020 P$(1)="PAPER      ";P$(2)="ROCK      "
1030 P$(3)="SCISSORS"
1040 C$(1)=P$(1);C$(2)=P$(2);C$(3)=P$(3)
1050 REM *** INSTRUCTIONS ***
1060 PRINTTAB(9)"-PAPER/ROCK/SCISSORS-"
1070 PRINTCHR$(17)CHR$(17)
1080 PRINT"THESE ARE THREE POINTS TO REMEMBER:--"
1090 PRINTTAB(10)"1) PAPER WRAPS ROCK"
1100 PRINTTAB(10)"2) ROCK BLUNTS SCISSORS"
1110 PRINTTAB(10)"3) SCISSORS CUT PAPER"
1120 PRINTCHR$(17)CHR$(17)
1130 PRINTTAB(9)"PRESS KEY'S P,R OR S"
1140 PRINTTAB(6)"FOR PAPER,ROCK OR SCISSORS"
1150 PRINTCHR$(17)CHR$(17)
1160 PRINTTAB(9)"PRESS A KEY TO PLAY"
1170 GETA$;IFA$=""THEN1170
1180 PRINTCHR$(147)
1190 REM *** PLAYERS CHOICE ***
1200 PRINTCHR$(19)"COMPUTER:"CS
1210 PRINTCHR$(19)TAB(20)"PLAYER:"PS
1220 PRINTCHR$(17)CHR$(17)
1230 PRINT"P-(PAPER),R-(ROCK)";
1240 PRINT"OR S-(SCISSORS)"
1250 PRINTCHR$(17)CHR$(17)
1260 GETA$
1270 IFA$<"P"AND A$<"R"AND A$<"S"THEN1260
1280 IFA$="P"THENGOSUB1430
1290 IFA$="R"THENGOSUB1440
1300 IFA$="S"THENGOSUB1450
1310 REM *** COMPUTERS CHOICE ***
1320 R=INT(RND(TI)*3+1)
1330 ONRGOSUB1470,1480,1490
1340 REM *** WHO SCORES ? ***
1350 IFPC=1ANDCC=2THENPS=PS+1
1360 IFPC=1ANDCC=3THENC$=CS+1

```



```
1370 IFPC=2ANDCC=1THENCS=CS+1
1380 IFPC=2ANDCC=3THENPS=PS+1
1390 IFPC=3ANDCC=1THENPS=PS+1
1400 IFPC=3ANDCC=2THENCS=CS+1
1410 GOTO1200
1420 REM *** PLAYER ***
1430 PRINTTAB(20);P$(1);PC=1:RETURN
1440 PRINTTAB(20);P$(2);PC=2:RETURN
1450 PRINTTAB(20);P$(3);PC=3:RETURN
1460 REM *** COMPUTER ***
1470 PRINTCHR$(145);C$(1);CC=1:RETURN
1480 PRINTCHR$(145);C$(2);CC=2:RETURN
1490 PRINTCHR$(145);C$(3);CC=3:RETURN
```

Searchlight



Your base is undergoing a night-time air raid. Use the searchlight to pick out the enemy bombers and then blast them out of the sky! There is no moon to help you but you do have three flares at your disposal if you cannot find your target. Shoot down as many aircraft as you can before your base is bombed-out.

Use the joystick to control the searchlight and shoot down aircraft. Flares are fired using the SPACE bar on your keyboard.

```

1000 POKE53281,0:POKE53280,0
1010 PRINTCHR$(147):PRINTCHR$(158)
1020 PRINTTAB(5)"          | | | |"
1030 PRINTTAB(5)"  | | | |"
1040 PRINTTAB(5)" | | | |"
1050 PRINTTAB(5)" | | | |"
1060 PRINTTAB(5)" | | | |"
1070 PRINTTAB(5)" | | | |"
1080 PRINTTAB(5)"  | | | | | | | |"
1090 PRINTTAB(5)"  | | | | | | | |"
1100 PRINTTAB(5)" | | | | | | | |"
1110 PRINTTAB(5)" | | | | | | | |"
1120 PRINTTAB(5)" | | | | | | | |"
1130 PRINTTAB(5)" | | | | | | | |"
1140 PRINTTAB(5)"          |"
1150 PRINTTAB(5)"          |"
1160 PRINTTAB(5)"          |"
1170 PRINTTAB(5)"          |"
1180 PRINTTAB(5)"          |"
1190 PRINTTAB(5)"          |"
1200 PRINT:PRINT"<<PRESS ANY KEY TO PLAY>>"
1210 DIMX(6),D(6),P(6)
1220 V=53248:FORI=1TO5:D(I)=1:P(I)=14:NEXT
1230 FL=3:S=54272
1240 GOSUB2520:REM M/C JOYSTICK
1250 GETA$:IFA$=""THEN1250
1260 GETJ$:IFJ$<>""THEN1260
1270 GOSUB2160:REM SET UP SCREEN
1280 REM----READ SPRITE DATA----
1290 FORI=12288TO12350:READA:POKEI,A:NEXT
1300 FORI=12352TO12414:READA:POKEI,A:NEXT
1310 FORI=896TO958:READA:POKEI,A:NEXT
1320 FORI=12672TO12734:READA:POKEI,A:NEXT
1330 FORI=12736TO12798:READA:POKEI,A:NEXT
1340 REM----COLOR SPRITES----
1350 REM--SPRITE 0--

```

Listing continued next page

```

1360 POKEV+39,2:POKEV+37,7:POKEV+38,1
1370 REM---SPRITE 1-5---
1380 FORI=V+40TOV+44:POKEI,0:NEXT
1390 REM---SPRITE 6---
1400 POKEV+45,7
1410 REM---SPRITE 7---
1420 POKEV+46,2
1430 POKEV+28,128:REM SET MULTI COLOR
1440 REM-----SET POINTERS-----
1450 POKE2040,192:REM AIR EXP
1460 POKE2046,198:REM SEARCHLIGHT
1470 POKE2047,199:REM GROUND EXP
1480 REM-----JOYSTICK LIMITS-----
1490 POKE49152,0:POKE49153,0:REM X2 MIN
1500 POKE49156,113:POKE49157,1:REM X2 MAX
1510 POKE49160,0:POKE49161,0:REM Y2 MIN
1520 POKE49164,192:POKE49165,0:REM Y2 MAX
1530 POKE49168,0:REM SCAN RATE
1540 SYS49183:REM ENTER WEDGE
1550 REM-----SET SEARCHLIGHT/DATA PRIORITY-----
1560 POKEV+27,64
1570 REM-----EXPAND SPRITES 6 & 7-----
1580 POKEV+23,192:POKEV+29,192
1590 REM-----INITIAL SEARCHLIGHT POSITION-----
1600 POKE49169,41:POKE49170,1
1610 POKE49173,129:POKE49174,0
1620 REM-----TURN ON SPRITE 6-----
1630 POKEV+21,64
1640 REM-----BOMBER FORMATION-----
1650 F=INT(RND(1)*3)
1660 IFF=0THEN GOSUB2940:REM FORMATION 1
1670 IFF=1THEN GOSUB2990:REM FORMATION 2
1680 IFF=2THEN GOSUB3040:REM FORMATION 3
1690 FORB=1TO5:POKEV+2*B,X(B):NEXT
1700 POKEV+3,Y1:POKEV+5,Y2:POKEV+7,Y3:POKEV+9,Y4:P
OKEV+11,Y5
1710 POKEV+9,Y4:POKEV+11,Y5

```

```

1720 POKEV+21,PEEK(V+21)OR62:REM TURN ON BOMBERS
1730 REM-----MAIN LOOP-----
1740 C=C+1
1750 GOSUB3090:REM MOVE SEARCHLIGHT
1760 FORB=1TO5
1770 IFX(B)>254THEND(B)=-1:P(B)=193
1780 IFX(B)<1THEND(B)=1:P(B)=14
1790 POKEV+2*B,X(B)
1800 POKE2040+B,P(B)
1810 GOSUB3090:REM MOVE SEARCHLIGHT
1820 X(B)=X(B)+D(B):NEXT
1830 IFC/20=INT(C/20)THEN GOSUB3440
1840 IFEN=1THEN1890:REM CHECK FOR END
1850 IF(PEEK(V+21)AND62)=0THEN1640
1860 REM---FLARES---
1870 GETF$:IFF$=" "ANDFL<>0THENGOSUB4090
1880 GOTO1730:REM RESTART LOOP
1890 REM-----END OF GAME-----
1900 PRINTCHR$(147):REM CLEAR SCREEN
1910 POKEV+21,0:REM TURN OFF SPRITES
1920 REM---END MESSAGE---
1930 POKE53281,0:POKE53280,0
1940 PRINTTAB(3)"  _  "
1950 PRINTTAB(3)" | |      |      |  | "
| "
1960 PRINTTAB(3)" | |      |      |  | | "
| "
1970 PRINTTAB(3)" | |      |      |  | |  | "
| "
1980 PRINTTAB(3)" | |      |      |  | |  | "
| "
1990 PRINTTAB(3)" | |      |      |  | |  | "
| "
2000 PRINTTAB(3)" | |      |      |  | |  | "
| "
2010 PRINTTAB(3)" H o m e b o y  |  |  F

```

Listing continued next page

```

|"
2020 PRINTTAB(3)" |  ||  |||||  ||  ||  |  |  ||  ||  |
|"
2030 PRINTTAB(3)" |  ||  |||||  ||  ||  |  |  ||  ||  |
|"
2040 PRINTTAB(3)" |  ||  |||||  |P|  |  |  |  ||  ||  |
|"
2050 PRINTTAB(3)" |  ||  ||  ||  ||  |  |  |  ||  ||  ||  |
|"
2060 PRINTTAB(3)" |  ||  ||  ||  ||  ||  |  |  ||  ||  ||  |
|"
2070 PRINTTAB(3)" |  ||  ||  ||  ||  ||  |  |  ||  ||  ||  |
"
2080 PRINTTAB(3)" |P|  |P|  |P|  |P|  |P|  |P|  |P|
0"
2090 PRINT:PRINTTAB(8)"YOUR SCORE  ";YS
2100 PRINT:PRINTTAB(5)CHR$(129);"<<PRESS ANY KEY T
O PLAY>>"
2110 SYS49208:REM DISABLE WEDGE
2120 REM---RESET SEARCHLIGHT POSITION---
2130 POKEV+12,41:POKEV+16,1:POKEV+13,129
2140 RUN1210
2150 END
2160 REM---SET UP THE SCREEN-----
2170 PRINTCHR$(147):REM CLEAR SCREEN
2180 POKE53280,6:POKE53281,0
2190 PRINTTAB(6)CHR$(145);CHR$(158)"YOUR SCORE 0"
2200 PRINTTAB(25)CHR$(145)"FLARES";FL
2210 REM---GROUND---
2220 FORI=1944TO2023:POKEI,160
2230 POKEI+54272,11:NEXT
2240 REM---TOWER---
2250 FORI=1908TO1708STEP-40
2260 POKEI,106:POKEI+1,86:POKEI+2,116
2270 POKEI+54272,11:POKEI+1+54272,11
2280 POKEI+2+54272,11:NEXT
2290 POKE1629,66:POKE1629+54272,11

```

```

2300 POKE1669,66:POKE1669+54272,11
2310 REM-----BUILDING ONE-----
2320 FORI=1920TO1923:POKEI,160
2330 POKEI-40,160:POKEI+54272,11:POKEI-40+54272,11
:NEXT
2340 POKE1879,233:POKE1884,223
2350 POKE1879+54272,11:POKE1884+54272,11
2360 REM--BUILDING TWO--
2370 FORI=1929TO1933:POKEI,160
2380 POKEI+54272,11:NEXT
2390 POKE1889,160:POKE1889+54272,11
2400 POKE1890,160:POKE1890+54272,11
2410 REM--OIL DRUMS--
2420 FORI=1939TO1943:POKEI,81
2430 POKEI+54272,11:NEXT
2440 FORI=1900TO1902:POKEI,81
2450 POKEI+54272,11:NEXT
2460 REM--STARS--
2470 FORI=1TO30
2480 P=INT(RND(1)*560)
2490 POKE1064+P,46:POKE1064+54272+P,1
2500 NEXT
2510 RETURN
2520 REM-----B/L FOR M/C JOYSTICK-----
2530 DATA120,173,20,3,141,29,192,173,21
2540 DATA3,141,30,192,169,71,141,20,3
2550 DATA169,192,141,21,3,88,96,120,173
2560 DATA29,192,141,20,3,173,30,192,141
2570 DATA21,3,88,96,173,27,192,205,16
2580 DATA192,240,3,76,65,193,169,255
2590 DATA141,27,192,162,0,160,0,185,2
2600 DATA220,141,28,192,169,224,153,2
2610 DATA220,185,0,220,72,173,28,192
2620 DATA153,2,220,189,8,192,221,21,192
2630 DATA189,9,192,253,22,192,144,18
2640 DATA189,8,192,157,21,192,189,9,192

```

Listing continued next page

```

2650 DATA157,22,192,104,74,72,76,162
2660 DATA192,104,74,72,176,13,222,21
2670 DATA192,189,21,192,201,255,208,3
2680 DATA222,22,192,189,21,192,221,12
2690 DATA192,189,22,192,253,13,192,144
2700 DATA18,189,12,192,157,21,192,189
2710 DATA13,192,157,22,192,104,74,72,76
2720 DATA207,192,104,74,72,176,8,254,21
2730 DATA192,208,3,254,22,192,189,0,192
2740 DATA221,17,192,189,1,192,253,18
2750 DATA192,144,18,189,0,192,157,17
2760 DATA192,189,1,192,157,18,192,104
2770 DATA74,72,76,1,193,104,74,72,176
2780 DATA13,222,17,192,189,17,192,201
2790 DATA255,208,3,222,18,192,189,17
2800 DATA192,221,4,192,189,18,192,253,5
2810 DATA192,144,18,189,4,192,157,17
2820 DATA192,189,5,192,157,18,192,104
2830 DATA74,72,76,46,193,104,74,72,176
2840 DATA8,254,17,192,208,3,254,18,192
2850 DATA104,74,176,5,169,1,153,25,192
2860 DATA200,232,232,224,4,240,3,76,91
2870 DATA192,238,27,192,108,29,192
2880 DATA35994:REM*CHECKSUM*
2890 FORI=49183TO49478
2900 READX:POKEI,X:CC=CC+X
2910 NEXT
2920 READX:IFX<>CCTHEN PRINT"CHECKSUM ERROR."
2930 RETURN
2940 REM-----FORMATION 1-----
2950 X(1)=0:Y1=115:X(2)=96:Y2=90
2960 X(3)=24:Y3=75:X(4)=56:Y4=60
2970 X(5)=144:Y5=110
2980 RETURN
2990 REM-----FORMATION 2-----
3000 X(1)=0:Y1=58:X(2)=48:Y2=75
3010 X(3)=36:Y3=90:X(4)=70:Y4=60

```



```

3020 X(5)=96:Y5=90
3030 RETURN
3040 REM-----FORMATION 3-----
3050 X(1)=0:Y1=60:X(2)=48:Y2=90
3060 X(3)=96:Y3=120:X(4)=144:Y4=156
3070 X(5)=192:Y5=170
3080 RETURN
3090 REM---MOVE SEARCHLIGHT---
3100 X6=PEEK(49169)+256*PEEK(49170)
3110 Y6=PEEK(49173)+256*PEEK(49174)
3120 H6=INT(X6/256):L6=X6-256*H6
3130 IFH6=1THENPOKEV+16,PEEK(V+16)OR65:GOTO3150
3140 POKEV+16,PEEK(V+16)AND190
3150 POKEV+12,L6:POKEV+13,Y6
3160 REM---FIRE BUTTON---
3170 POKE49177,0
3180 FB2= PEEK(49177)
3190 IFFB2=1THENGOSUB3210:REM FIRE
3200 RETURN
3210 REM-----FIRE ROUTINE-----
3220 POKE(V+30),0:REM RESET COLL REGISTER
3230 L0=L6+12
3240 IFL0>255THENPOKEV+16,PEEK(V+16)OR1:L0=L0-255
3250 PV=PEEK(V+21)
3260 POKEV+21,PVOR1
3270 POKEV,L0:POKEV+1,Y6+11:FORI=1TO20:NEXT
3280 GOSUB3330
3290 POKE(V+21),PVAND(255-(PEEK(V+30)AND63))
3300 IFPEEK(V+21)<PVTHENYS=YS+100:PRINTTAB(6)CHR$(
145);"YOUR SCORE":YS:C=C-5
3310 IFC<0THENC=1
3320 RETURN
3330 REM---SOUND EFFECT #2---
3340 S=54272
3350 POKES+24,15
3360 POKES+6,128

```

Listing continued next page

```

3370 POKES+4,129
3380 POKES,75:POKES+1,3
3390 POKES+5,131
3400 FORI=15TO0STEP-.4
3410 POKES+24,I
3420 NEXT
3430 RETURN
3440 REM-----GROUND ATTACK-----
3450 POKEV+15,192:REM SET Y COORD
3460 POKEV+16,PEEK(V+16)AND127
3470 A=A+1:ONAGOTO3480,3520,3560,3610,3650,3700,37
40,3790,3850
3480 GOSUB3920
3490 POKEV+14,20:POKEV+21,PEEK(V+21)OR128:FORI=1TO
70:NEXT
3500 POKEV+21,PEEK(V+21)AND127
3510 GOSUB4020:GOTO3910
3520 GOSUB3920
3530 POKEV+14,92:POKEV+21,PEEK(V+21)OR128:FORI=1TO
70:NEXT
3540 POKEV+21,PEEK(V+21)AND127
3550 GOSUB4020:GOTO3910
3560 GOSUB3920
3570 POKEV+14,132:POKEV+21,PEEK(V+21)OR128:FORI=1T
070:NEXT
3580 FORI=1919TO1921:POKEI,32:POKEI-40,32:NEXT
3590 POKEV+21,PEEK(V+21)AND127
3600 GOSUB4020:GOTO3910
3610 GOSUB3920
3620 POKEV+14,252:POKEV+21,PEEK(V+21)OR128:FORI=1T
070:NEXT
3630 POKEV+21,PEEK(V+21)AND127
3640 GOSUB4020:GOTO3910
3650 GOSUB3920
3660 POKEV+14,44:POKEV+21,PEEK(V+21)OR128:FORI=1TO
70:NEXT
3670 FORI=1629TO1909STEP40:POKEI,32:POKEI-1,32:POK

```

```

EI+1,32:NEXT
3680 POKEV+21,PEEK(V+21)AND127
3690 GOSUB4020:GOTO3910
3700 GOSUB3920
3710 POKEV+14,188:POKEV+21,PEEK(V+21)OR128:FORI=1T
070:NEXT
3720 POKEV+21,PEEK(V+21)AND127
3730 GOSUB4020:GOTO3910
3740 GOSUB3920
3750 POKEV+14,220:POKEV+21,PEEK(V+21)OR128:FORI=1T
070:NEXT
3760 FORI=1929TO1933:POKEI,32:POKEI-40,32:NEXT
3770 POKEV+21,PEEK(V+21)AND127
3780 GOSUB4020:GOTO3910
3790 GOSUB3920
3800 POKEV+14,37:POKEV+16,PEEK(V+16)OR128
3810 POKEV+21,PEEK(V+21)OR128:FORI=1TO70:NEXT
3820 FORI=1939TO1943:POKEI,32:POKEI-40,32:NEXT
3830 POKEV+21,PEEK(V+21)AND127
3840 GOSUB4020:GOTO3910
3850 GOSUB3920
3860 POKEV+14,156:POKEV+21,PEEK(V+21)OR128:FORI=1T
0200:NEXT
3870 FORI=1922TO1924:POKEI,32:POKEI-40,32:NEXT
3880 POKEV+21,PEEK(V+21)AND127
3890 GOSUB4020
3900 EN=1
3910 RETURN
3920 REM-----SOUND EFFECT-----
3930 S=54272
3940 POKES+5,96:POKES+4,17:POKES+24,15
3950 POKES+6,128
3960 FORI=50000TO14000STEP-500
3970 HF=INT(I/256):LF=I-256*HF
3980 POKES,LF:POKES+1,HF
3990 NEXT

```

Listing continued next page

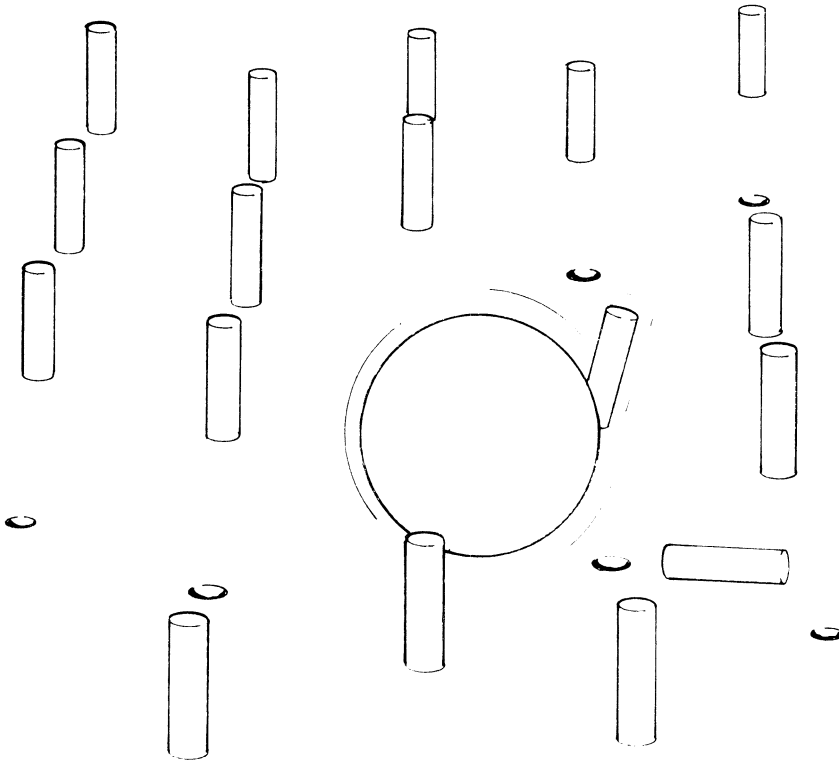
```

4000 POKES+24,0
4010 RETURN
4020 REM-----EXPLOSION-----
4030 POKES+4,129:POKES,127:POKES+1,5
4040 FORI=15TO0STEP-.05
4050 POKES+24,I
4060 NEXT
4070 RETURN
4080 END
4090 REM-----FLARES-----
4100 GOSUB3920:REM SOUND
4110 FL=FL-1:IFFL<0THENFL=0
4120 FORI=1TO10
4130 POKE53281,6:FORJ=1TO50:NEXT
4140 POKE53281,1:FORJ=1TO50:NEXT
4150 NEXT
4160 POKE53281,0
4170 PRINTTAB(25)CHR$(145)"FLARES":FL
4180 RETURN
4190 REM-----DATA FOR AIR EXP-----
4200 DATA0,0,0,0,0,0,0,16,0,0,8,0,4,16
4210 DATA0,3,2,64,1,56,128,12,255,144
4220 DATA1,238,40,5,151,0,11,121,0,1
4230 DATA183,0,25,214,96,0,236,48,6,24
4240 DATA152,3,98,0,8,51,0,0,96,128,0
4250 DATA64,0,0,0,0,0,0,0
4260 REM-----DATA FOR LEFT BOMBER-----
4270 DATA0,0,0,0,0,0,0,0,0,0,0,0,0,0
4280 DATA14,0,0,14,0,0,30,15,255,254
4290 DATA241,255,126,254,219,247,127
4300 DATA255,128,3,252,0,0,254,0,0,63
4310 DATA0,0,0,0,0,0,0,0,0,0,0,0
4320 DATA0,0,0,0,0,0,0,0,0,0,0,0
4330 REM-----DATA FOR RIGHT BOMBER-----
4340 DATA0,0,0,0,0,0,0,0,0,0,0,0
4350 DATA112,0,0,112,0,0,120,0,0,127
4360 DATA255,240,126,255,143,239,219

```

4370 DATA127,1,255,254,0,63,192,0,127
4380 DATA0,0,252
4390 DATA0,0,0,0,0,0,0,0,0,0
4400 DATA0,0,0,0,0,0,0,0,0,0
4410 REM----DATA FOR SEARCHLIGHT----
4420 DATA0,0,0,0,126,0,1,255,128,7,255
4430 DATA224,15,255,240,31,255,248,31
4440 DATA255,248,63,255,252,63,255,252
4450 DATA127,255,254,127,255,254,63,255
4460 DATA252,63,255,252,31,255,248,31
4470 DATA255,248,15,255,240,7,255,224,1
4480 DATA255,128,0,126,0,0,0,0,0,0
4490 REM----DATA FOR GROUND EXP----
4500 DATA0,0,0,0,0,0,0,0,0,0,24,0,0,24
4510 DATA0,0,48,0,0,170,0,0,68,0,8,110
4520 DATA0,4,176,192,0,166,128,3,105
4530 DATA128,0,51,0,5,60,0,1,154,128,2
4540 DATA124,0,0,186,0,0,92,0,0,56
4550 DATA0,0,60,0,0,24,0,0

Flip Flap



Flip Flap is an original game based on an old idea. Can you remember those games you used to play in the arcades before Space Invaders came along? You can't ??? ... Well, there was a game where you drop a coin in the top of a box and it would gradually work its way down to the bottom, randomly bouncing off a grid of pins, determining which way the coin would travel and once at the bottom ... I'm afraid that I am completely oblivious to its aim, but it seemed a good foundation for a game. So Flip Flap was born.

The idea of this game is more apparent. You have a continuously bouncing ball at the top of the screen and when this reaches one side it rebounds the other way. At this point you can, at any time, press a 'letter' key on the keyboard and the ball will start to drop down the screen.

As the ball drops, it collides with pins that are placed down the screen, and each time it hits a pin, the pin gets knocked in and you score some points, with a bonus if you can hit ten pins on each drop (not impossible). As the game progresses so it gets a little harder. If you hit a pin that has already been knocked in you score no points. Sorry!

The game ends when you have dropped 25 balls (or sooner if you are a bad loser!).

```

1000 REM *** FLIP FLAP ***
1010 GOSUB1760:REM TITLE
1020 GOSUB1620:REM DRAW SCREEN
1030 S=54272:REM START OF 'SID' CHIP
1040 FORI=0TO24:POKES+I,0:NEXT:REM CLEAR
1050 POKES+5,8:POKES+22,74:POKES+23,1
1060 POKES+24,79
1070 REM *** SET UP VARIABLES ***
1080 X=1:Y=1:D=1:SC=1024:CO=55296
1090 REM *** MAIN LOOP ***
1100 REM *** DROP ROUTINE ***
1110 OX=X:OY=Y:REM OLD POS. X & Y
1120 HT=0:IFB=25THEN1460
1130 X=X+D:IFX=0ORX>=39THEND=-D
1140 POKESC+OX+40*OY,32
1150 POKECO+X+40*Y,1:POKESC+X+40*Y,81
1160 GETA$:IFA$=""THEN1110
1170 REM *** MAIN ROUTINE ***
1180 B=B+1
1190 OX=X:OY=Y:Y=Y+1
1200 IFY>23THENGOSUB1740:GOTO1110
1210 PRINTCHR$(19)CHR$(5)"SCORE:"SE
1220 PRINTCHR$(19)TAB(15)"DROPS:"B
1230 IFSE>HITHENHI=SE
1240 PRINTCHR$(19)TAB(30)"HIGH:"HI
1250 P=PEEK(SC+X+40*Y)
1260 IFP=930RP=46THENZ=RND(1):GOSUB1320
1270 IFP=93THENSE=SE+1
1280 POKESC+OX+40*OY,32
1290 POKECO+X+40*Y,1:POKESC+X+40*Y,81
1300 GOTO1190:REM END LOOP
1310 REM *** CHANGE DIRECTION ***
1320 POKECO+X+40*Y,1:POKESC+X+40*Y,46
1330 X=OX:Y=OY
1340 IFZ<.5THENX=X-1:IFX=0THENX=39
1350 IFZ>.5THENX=X+1:IFX=39THENX=0

```

Listing continued next page

```

1360 IFP=46THENGOSUB1430:RETURN
1370 POKES+0,33:POKES+1,INT(RND(1)*50+9)
1380 POKES+4,128:FORT=1TO20:NEXT
1390 HT=HT+1
1400 IFHT=10THENSE=SE+INT(RND(1)*9):HT=0
1410 POKES+4,129:RETURN
1420 REM *** WHOOPS! ***
1430 POKES+0,90:POKES+1,50
1440 POKES+4,32:FORT=1TO20:NEXT
1450 POKES+4,33:RETURN
1460 REM *** SORRY! GAME OVER ***
1470 FORT=1TO25:PRINTCHR$(19)CHR$(17);
1480 PRINTCHR$(157)CHR$(148):POKE218,160
1490 NEXTT
1500 POKES+4,0:POKES+5,31:POKES+6,240
1510 PRINTCHR$(17)CHR$(17)CHR$(17)
1520 PRINTTAB(11)"-SORRY! GAME OVER-"
1530 POKES+4,33:FORT=1TO100
1540 R=INT(RND(TI)*10+1):POKE53280,R
1550 POKES+1,1+T+R:POKES,200
1560 FORK=1TO20:NEXT
1570 NEXTT:POKES+4,0:POKE53280,0
1580 PRINTCHR$(17)TAB(6);
1590 PRINT"-PRESS A KEY TO PLAY AGAIN-"
1600 GETA$:IFA$=""THEN1600
1610 SE=0:B=0:GOTO1010:REM PLAY AGAIN
1620 REM *** DRAW SCREEN ***
1630 PRINTCHR$(147)
1640 POKE53280,0:POKE53281,0
1650 PRINTCHR$(17)
1660 PRINTCHR$(158):FORT=1TO5
1670 PRINT" | | | | | | | | | | ";
1680 PRINT" | | | | | | | | | | "
1690 PRINT" | | | | | | | | | | ";
1700 PRINT" | | | | | | | | | | "
1710 NEXT
1720 RETURN

```

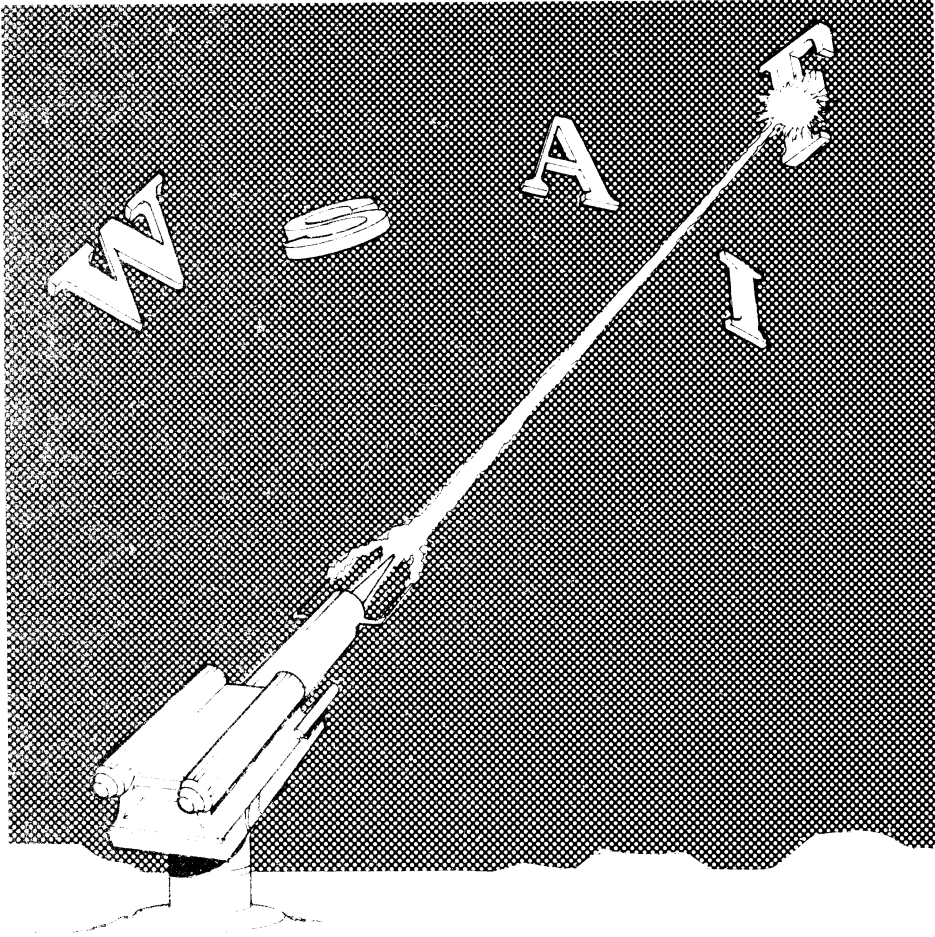


```

1730 REM *** POKE SPACE ***
1740 POKE53280+0X+40*0Y,32
1750 Y=1:X=1:RETURN
1760 REM *** TITLE PAGE ***
1770 PRINTCHR$(147)CHR$(156)
1780 POKE53280,0:POKE53281,0
1790 PRINTTAB(5)" "
1800 PRINTTAB(5)"  \ "
1810 PRINTTAB(5)" | "
1820 PRINTTAB(5)" || 000 0 0 000 "
1830 PRINTTAB(5)" || 0 0 0 0 0 "
1840 PRINTTAB(5)" || 00 0 0 000 "
1850 PRINTTAB(5)" || 0 0 0 0 "
1860 PRINTTAB(5)" || 0 000 0 0 "
1870 PRINTTAB(5)" \ "
1880 PRINTTAB(17)" "
1890 PRINTTAB(17)"  \ "
1900 PRINTTAB(17)" | "
1910 PRINTTAB(17)" || 000 0 0 000 "
1920 PRINTTAB(17)" || 0 0 0 0 0 0 "
1930 PRINTTAB(17)" || 00 0 000 000 "
1940 PRINTTAB(17)" || 0 0 0 0 0 "
1950 PRINTTAB(17)" || 0 000 0 0 0 "
1960 PRINTTAB(17)" \ "
1970 PRINTCHR$(17)CHR$(17)
1980 PRINTTAB(8)"-PRESS A KEY TO PLAY-"
1990 GETA$:IFA$=""THEN1990
2000 RETURN

```

Laser Spell



Spell the word displayed at the top by hitting the letters as they appear on the screen. Use the joystick to move the laser base and fire. The program chooses a word at random from those in the DATA statements at the end. Of course, you can change these words if you wish to make the game easier or more difficult.

(Educational Space Invaders??!)

```

1000 REM *** LASER SPELL ***
1010 POKE53280,0:POKE53281,0
1020 DIML$(20)
1030 FORI=1TO20:READL$(I):NEXT
1040 S=54272:REM SOUND
1050 FORI=0TO24:POKE5+I,0:NEXT
1060 POKE5+24,14
1070 POKE5+5,31:POKE5+6,240:GOSUB1670
1080 PRINTCHR$(147)
1090 POKE53280,0:POKE53281,0
1100 REM *** SCREEN/COLOR BASE ***
1110 SC=1024:CO=55296
1120 X=19:Q=-1:J=0
1130 P=INT(RND(1)*20)
1140 W$=L$(P)
1150 B$="┐└":REM BASE
1160 W=LEN(W$)
1170 PRINTCHR$(19)CHR$(8);W$
1180 POKE781,23:POKE782,0:SYS65520
1190 FORI=1TO40:PRINT"";:NEXT
1200 REM *** MAIN LOOP ***
1210 P=PEEK(56320):REM PORT 2
1220 IFP=123THENX=X-1:IFX<0THENX=0
1230 IFP=119THENX=X+1:IFX>35THENX=35
1240 IFP=111THENGOSUB1330
1250 IFJ=WTHENGOSUB1670:GOTO1080:REM END GAME
1260 POKE781,22:POKE782,X:SYS65520
1270 PRINT" ";B$;" "
1280 Q=Q+1
1290 IFQ=0THENGOSUB1430:REM PLOT LETTER
1300 IFQ>50THENGOSUB1480:REM CLR LETTER
1310 GOTO1210
1320 REM *** FIRE LASER BOLT ***
1330 B0=X+2:FORI=1TO20:Q=Q+1
1340 IFQ>50THENGOSUB1480:R=0
1350 POKE781,22-I:POKE782,B0:SYS65520

```

Listing continued next page

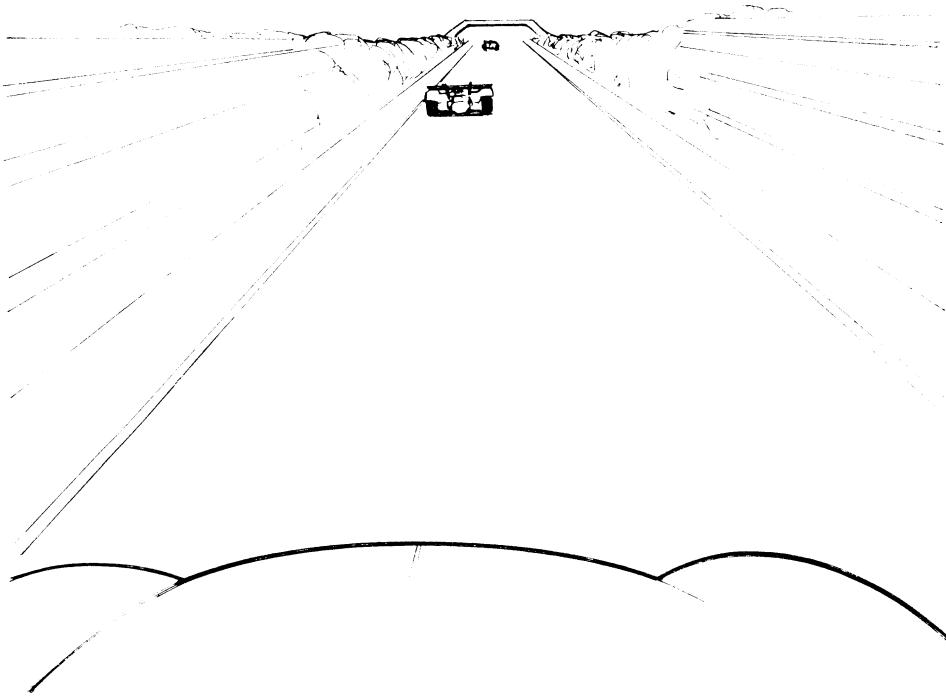
```

1360 PRINT"|"
1370 IFPEEK(SC+R)=93THENGOSUB1510:I=20
1380 POKE781,22-I:POKE782,B0:SYS65520
1390 PRINT" ":NEXT
1400 IFR=0THENQ=-1
1410 RETURN
1420 REM *** PLOT RND LETTER ***
1430 R=INT(RND(1)*520+120)
1440 L=INT(RND(1)*26+1)
1450 POKECO+R,7:POKESC+R,L
1460 Q=0:RETURN
1470 REM *** CLEAR RND LETTER ***
1480 POKESC+R,32
1490 Q=-1:RETURN
1500 REM *** LETTER HIT ***
1510 POKES+4,33
1520 POKESC+R,46:POKESC+R,86
1530 POKES,200:POKES+1,100
1540 FORT=1TO50:NEXT:POKESC+R,32
1550 POKES+4,0
1560 FORT=1TOLEN(W#)
1570 IFL=PEEK(SC+T-1)THENGOSUB1610
1580 NEXT
1590 RETURN
1600 REM *** INVERT LETTER ***
1610 REM
1620 POKESC+T-1,PEEK(SC+T-1)+128
1630 T=LEN(W#)
1640 J=J+1
1650 RETURN
1660 REM *** END GAME ROUTINE ***
1670 PRINTCHR$(147)
1680 POKES+4,33
1690 FORT=10TO50
1700 POKES,100:POKES+1,T
1710 POKE646,INT(RND(1)*15)
1720 PRINTTAB(7)"PRESS A KEY TO PLAY AGAIN":NEXTT

```

```
1730 POKES+4,0
1740 GETA$:IFA$="" THEN 1740
1750 RETURN
1760 REM *** WORD DATA ***
1770 DATA"TELEVISION","VEGETABLE"
1780 DATA"COMMODORE","COMPUTER"
1790 DATA"ALUMINIUM","DICTIONARY"
1800 DATA"PENCIL","SIDEBOARD"
1810 DATA"JOYSTICK","GUITAR"
1820 DATA"KEYBOARD","SPAGHETTI"
1830 DATA"TRANSISTOR","SPACESHIP"
1840 DATA"CHRISTMAS","BIRTHDAY"
1850 DATA"PROGRAM","ARCADE"
1860 DATA"CHAIR","BICYCLE"
```

Track Racer



In Track Racer you have to travel down a dirt track that gradually gets narrower. Each time you score a hundred points the width of the track is reduced by one character space. If you crash into one of the posts along the edge your car will explode into an array of color.

The car is moved from side to side using the joystick; but be careful because unless you steer the other way you will continue to drift in towards the kerb!

This program makes use of a short routine that causes the screen to scroll downwards. This is slightly more difficult than causing the screen to scroll up but the visual effect is well worth the extra effort required.

```

1000 REM *** TRACK RACER ***
1010 PRINTCHR$(147)
1020 POKE53280,0:POKE53281,0
1030 GOSUB1720:REM TITLE
1040 REM *** READ SPRITE DATA ***
1050 FORI=0TO62:READJ:POKE832+I,J:NEXT
1060 FORI=0TO62:READJ:POKE896+I,J:NEXT
1070 REM *** SET UP SPRITES ***
1080 POKE2040,13:VC=53248:POKEVC+21,1
1090 POKE2041,14:POKEVC+40,8
1100 POKEVC+39,14:POKEVC+23,3:POKEVC+29,3
1110 POKEVC+0,150:POKEVC+1,180
1120 POKEVC+28,2
1130 REM *** SET UP SOUND ***
1140 S=54272:FORI=0TO24:POKES+I,0:NEXT
1150 POKES+5,31:POKES+6,240
1160 POKES+24,15
1170 L=12:X=150:W=12
1180 POKE646,14
1190 PRINTCHR$(147):SE=0
1200 FORI=1TO20
1210 PRINTTAB(L)"  "SPC(W)"  "
1220 NEXT:POKEVC+31,0
1230 POKES+4,33:POKES,100:POKES+1,4
1240 PRINTCHR$(19)CHR$(158)"SCORE:"
1250 PRINTCHR$(19)TAB(28)"HIGH:"
1260 REM *** MAIN LOOP ***
1270 DIR=1:IFRND(1)<.5THENDIR=2
1280 SE=SE+1:Q=Q+1
1290 IFDIR=1THENL=L-1
1300 IFL<10THENL=L+1
1310 IFDIR=2THENL=L+1
1320 IFL>20THENL=L-1
1330 PRINTCHR$(19)CHR$(17);
1340 PRINTCHR$(157)CHR$(148)
1350 POKE218,160

```

Listing continued next page

```

1360 PRINTTAB(L)"↑"SPC(W)"↑"
1370 P=PEEK(56320)
1380 IFP=123THENZ=1
1390 IFP=119THENZ=2
1400 ONZGOSUB1510,1530
1410 POKEVC+0,X
1420 PRINTCHR$(19)TAB(6);SE
1430 IFSE>HITHENHI=SE
1440 PRINTCHR$(19)TAB(34);HI
1450 IFQ=100THENW=W-1:Q=0
1460 IFPEEK(VC+31)AND1=1THEN1560
1470 POKEVC+31,0
1480 IFW=8THENW=12
1490 GOTO1270
1500 REM *** CAR DIRECTION ***
1510 X=X-4:IFX<10THENX=10
1520 RETURN
1530 X=X+4:IFX>250THENX=250
1540 RETURN
1550 REM *** CRASH INTO KERB ***
1560 POKEVC+21,2
1570 POKEVC+2,X:POKEVC+3,100
1580 FORV=15TO0STEP-.2:POKES+1,2+V
1590 POKES+4,129:POKES+24,V
1600 POKEVC+37,INT(RND(TI)*15)
1610 POKEVC+38,INT(RND(TJ)*15)
1620 NEXT:POKES+4,0
1630 POKEVC+21,0
1640 REM *** NEW GAME ***
1650 PRINTCHR$(19)
1660 POKE646,INT(RND(TI)*15)
1670 FORI=1TO4:PRINTCHR$(17):NEXT
1680 PRINTTAB(7)"PRESS FIRE ";
1690 PRINT"TO PLAY AGAIN"
1700 IFPEEK(56320)<>111THEN1650
1710 GOTO1080
1720 REM *** TITLE PAGE ***

```



```

1730 PRINTCHR$(19)CHR$(30)
1740 PRINTTAB(7)"< T R A C K * ";
1750 PRINT"R A C E R >"
1760 PRINTCHR$(17)CHR$(17)
1770 PRINTTAB(5)"USE JOYSTICK TO ";
1780 PRINT"STEER YOUR CAR"
1790 PRINTTAB(5)"ALONG THE SLIPPERY ";
1800 PRINT"TRACK....."
1810 PRINTTAB(5)"BUT WATCH OUT, ";
1820 PRINT"THE TRACK GETS"
1830 PRINTTAB(5)"NARROWER....."
1840 PRINTCHR$(17)CHR$(17)CHR$(17)
1850 PRINTTAB(9)"PRESS A KEY TO PLAY"
1860 GETA$:IFA$=""THEN1860
1870 RETURN:REM INTO GAME
1880 REM *** SPRITE 0 (CAR) ***
1890 DATA0,248,0,0,248,0,0,248,0,5,141
1900 DATA0,7,239,0,5,221,0,1,220,0,1
1910 DATA252,0,1,140,0,1,36,0,1,116,0,1
1920 DATA252,0,1,252,0,3,86,0,54,219,96
1930 DATA55,223,96,63,87,224,54,251,96
1940 DATA50,2,96,7,255,0,7,255,0
1950 REM *** SPRITE 1 (EXPLODE) ***
1960 DATA0,32,0,2,73,52,6,157,193,104
1970 DATA188,29,131,54,124,44,118,249
1980 DATA65,231,242,31,227,178,78,8,57
1990 DATA39,63,156,47,159,30,73,222,124
2000 DATA36,194,240,73,248,198,19,253
2010 DATA216,71,205,144,23,15,36,16,103
2020 DATA72,13,146,80,0,8,144,3,87,32

```

Mine Field



Minefield is a game of logic and strategy. You are in control of the mine detector and must destroy as many mines as possible in three minutes. Unfortunately, your assistant, Copy-Cat Kenneth mirrors your every move. Don't let him stand on a mine or you will both be blown to kingdom come!

To destroy the mines first position the detector over a mine then press the fire button. Each mine successfully destroyed scores you 100 points.

TREAD CAREFULLY!!!

```

1000 PRINTCHR$(147)
1010 GOSUB2570
1020 GOSUB1850:REM M/C CHAR-SHIFTER
1030 GOSUB2150:REM M/C JOYSTICK
1040 REM SPRITE NUMBER 0 AND 1
1050 REM STORED IN CASSETTE BUFFER
1060 POKE2040,13:POKE2041,14
1070 POKE53281,0
1080 REM-----RANDOM DIST OF MINES-----
1090 REM-----READ SPRITE DATA-----
1100 FOR I=832 TO 894:READA
1110 POKEI,A:NEXT
1120 FOR I=896TO958:READA
1130 POKEI,A:NEXT
1140 V=53248
1150 POKEV+39,2:REM COL SPR 0 RED
1160 POKEV+40,7:REM COL SPR 1 WHITE
1170 REM-----READ MINE DATA
1180 FORI=12504TO12519:READA
1190 POKEI,A:NEXT
1200 GETA$:IFA$=""THEN1200
1210 GETJ$:IFJ$<>""THEN1210
1220 PRINTCHR$(147)
1230 REM-----RANDOM DIST OF MINES-----
1240 FORI=1TO40:SC=INT(RND(1)*840)
1250 IFSC=380RSC=39THENI=I-1:GOTO1290
1260 IFSC/2<>INT(SC/2)THENI=I-1:GOTO1290
1270 POKE1104+SC,27:POKE1104+1+SC,28
1280 POKE55376+SC,5:POKE55376+1+SC,5
1290 NEXT
1300 REM-----JOYSTICK LIMITS-----
1310 POKE49152,24:POKE49153,0:REM X MIN
1320 POKE49156,70:POKE49157,1:REM X MAX
1330 POKE49160,60:POKE49161,0:REM Y MIN
1340 POKE49164,225:POKE49165,0:REM Y MAX
1350 POKE49168,1:REM SCAN RATE

```

Listing continued next page

```

1360 SYS49183:REM ENABLE WEDGE
1370 TI$="0000000";REM TIMER
1380 PRINTTAB(6)CHR$(145)CHR$(158)"YOUR SCORE 000"
1390 POKE49169,24:POKE49170,0:REM INITIAL X
1400 POKE49173,229:POKE49174,0:REM INITIAL Y
1410 POKEV,24:POKEV+1,229:POKEV+2,66:POKEV+16,2
1420 POKEV+3,50
1430 POKEV+21,3:REM TURN ON SPRITES 0 AND 1
1440 REM----MAIN MOVEMENT ROUTINE----
1450 POKEV+31,0:REM SET COLLISION REG
1460 PRINTCHR$(145);TAB(23)"TIME ";MID$(TI$,4,1);": ";RIGHT$(TI$,2)
1470 IFVAL(TI$)>=300THENGOSUB3070:REM END ROUTINE
1480 X=PEEK(49169)+256*PEEK(49170)
1490 Y=PEEK(49173)+256*PEEK(49174)
1500 N=345-X:M=279-Y
1510 HN=INT(N/256):LN=N-256*HN
1520 HX=INT(X/256):LX=X-256*HX
1530 IFX<90THEN POKEV+16,2:GOTO1550
1540 POKEV+16,HX
1550 POKEV,LX:POKEV+1,Y
1560 POKEV+2,LN:POKEV+3,M
1570 HC=PEEK(V+31):REM COLLISION DETECT
1580 REM--IF THERE IS MAN/MINE COLLISION--
1590 IFHC=2ORHC=3THENGOTO1680
1600 REM--IF FIRE BUTTON PRESSED--
1610 POKE49177,0
1620 FB2=PEEK(49177)
1630 IFFB2<>1THEN1450
1640 REM--IF THERE IS DET/MINE COLLISION--
1650 IFHCAND1=1THENGOSUB2870
1660 GOTO1450
1670 END
1680 REM----MINE/MAN COLLISION----
1690 PRINTCHR$(147)
1700 REM--FLASH SCREEN--
1710 FORI=1TO5

```

```

1720 GOSUB2760:REM FINAL EXPLOSION
1730 NEXT
1740 REM---SWITCH OFF SPRITES---
1750 POKEV+21,0
1760 FORI=1TO9:PRINTCHR$(17);:NEXT
1770 PRINTTAB(4)"YOU HAVE KILLED YOUR ASSISTANT!!"
1780 FORI=1TO3:PRINTCHR$(17);:NEXT
1790 PRINTTAB(12)"YOUR SCORE";YS
1800 FORI=1TO3:PRINTCHR$(17);:NEXT
1810 PRINTTAB(6)"<<<PRESS ANY KEY TO PLAY>>>"
1820 SYS49208:REM DISABLE WEDGE
1830 RUN1020
1840 END
1850 REM-----B/L FOR M/C CHAR-SHIFTER-----
1860 POKE52,48:POKE56,48:REM LOWER MEMTOP
1870 FORI=828TO950      :REM-TEMP STORE-
1880 READJ              :REM-----
1890 AA=AA+J            :REM-BASIC
1900 POKEI,J            :REM LOADER-
1910 NEXT               :REM-----
1920 READJ              :REM-CHECKSUM
1930 IF AA<>J THEN PRINT"DATA ERROR":END
1940 SYS(828):REM -EXECUTE-
1950 POKE53272,28:REM -POINT TO RAM-
1960 DATA169,0,141,14,220,169,51,141
1970 DATA1,0,162,0,189,0,208,157,0,48
1980 DATA189,0,209,157,0,49,189,0,210
1990 DATA157,0,50,189,0,211,157,0,51
2000 DATA189,0,212,157,0,52,189,0,213
2010 DATA157,0,53,189,0,214,157,0,54
2020 DATA189,0,215,157,0,55,232,224
2030 DATA255,208,203,173,255,208,141
2040 DATA255,48,173,255,209,141,255
2050 DATA49,173,255,210,141,255,50
2060 DATA173,255,211,141,255,51,173
2070 DATA255,212,141,255,52,173,255

```

Listing continued next page

```

2080 DATA213,141,255,58,173,255,214
2090 DATA141,255,54,169,55,141,1,0
2100 DATA169,1,141,14,220,169,28,141
2110 DATA24,208,96
2120 REM -CHECKSUM-
2130 DATA16246
2140 RETURN
2150 REM-----M/C JOYSTICK ROUTINE-----
2160 DATA120,173,20,3,141,29,192,173,21
2170 DATA3,141,30,192,169,71,141,20,3
2180 DATA169,192,141,21,3,88,96,120,173
2190 DATA29,192,141,20,3,173,30,192,141
2200 DATA21,3,88,96,173,27,192,205,16
2210 DATA192,240,3,76,65,193,169,255
2220 DATA141,27,192,162,0,160,0,185,2
2230 DATA220,141,28,192,169,224,153,2
2240 DATA220,185,0,220,72,173,28,192
2250 DATA153,2,220,189,8,192,221,21,192
2260 DATA189,9,192,253,22,192,144,18
2270 DATA189,8,192,157,21,192,189,9,192
2280 DATA157,22,192,104,74,72,76,162
2290 DATA192,104,74,72,176,13,222,21
2300 DATA192,189,21,192,201,255,208,3
2310 DATA222,22,192,189,21,192,221,12
2320 DATA192,189,22,192,253,13,192,144
2330 DATA18,189,12,192,157,21,192,189
2340 DATA13,192,157,22,192,104,74,72,76
2350 DATA207,192,104,74,72,176,8,254,21
2360 DATA192,208,3,254,22,192,189,0,192
2370 DATA221,17,192,189,1,192,253,18
2380 DATA192,144,18,189,0,192,157,17
2390 DATA192,189,1,192,157,18,192,104
2400 DATA74,72,76,1,193,104,74,72,176
2410 DATA13,222,17,192,189,17,192,201
2420 DATA255,208,3,222,18,192,189,17
2430 DATA192,221,4,192,189,18,192,253,5
2440 DATA192,144,18,189,4,192,157,17

```

```

2450 DATA192,189,5,192,157,18,192,104
2460 DATA74,72,76,46,193,104,74,72,176
2470 DATA8,254,17,192,208,3,254,18,192
2480 DATA104,74,176,5,169,1,153,25,192
2490 DATA200,232,232,224,4,240,3,76,91
2500 DATA192,238,27,192,108,29,192
2510 DATA35994:REM*CHECKSUM*
2520 FORI=49183TO49478
2530 READX:POKEI,X:CC=CC+X
2540 NEXT
2550 READX:IFX<>CCTHENPRINT"CHECKSUM ERROR"
2560 RETURN
2570 REM-----TITLE PAGE-----
2580 POKE53281,0:PRINTCHR$(30)
2590 PRINT:PRINT:PRINT
2600 PRINTTAB(18)"I"
2610 PRINTTAB(6)"M           F           D"
2620 PRINTTAB(7)"I           E"
2630 PRINTTAB(11)"E"
2640 PRINTTAB(9)"N"
2650 PRINTTAB(22)"L"
2660 PRINT:PRINT:PRINT
2670 PRINTTAB(30)"E"
2680 PRINTTAB(24)"Y   K"
2690 PRINTTAB(9)"R   S           A"
2700 PRINTTAB(12)"E   S           N           Y"
2710 PRINTTAB(7)"P"
2720 PRINT:PRINT
2730 PRINTTAB(24)"O"
2740 PRINTTAB(23)"T   PLAY"
2750 RETURN
2760 REM-----FINAL EXPLOSION-----
2770 IFI/2=INT(I/2)THENPOKE53281,2:GOTO2790
2780 POKE53281,7
2790 S=54296:W=54276:A=54277
2800 H=54273:L=54272

```

Listing continued next page

```

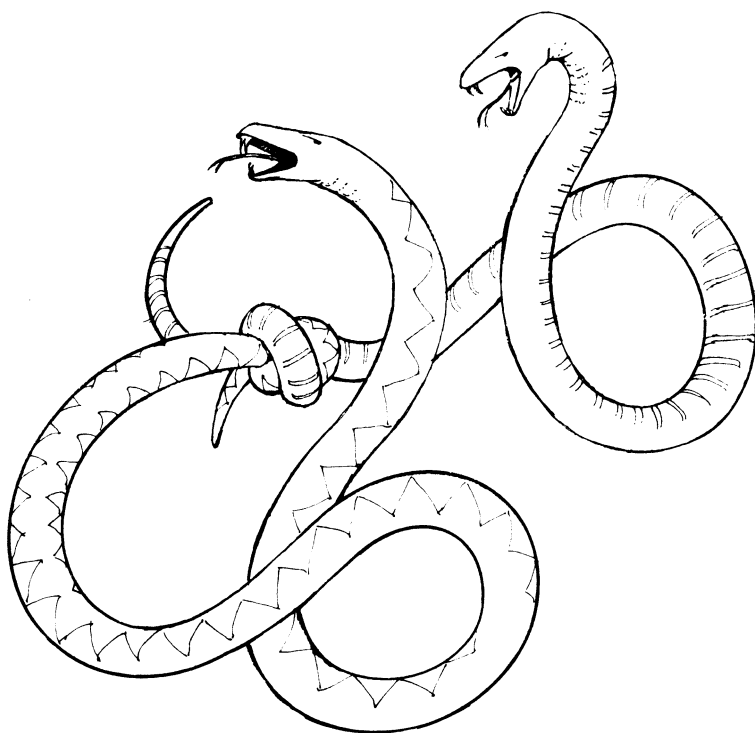
2810 FORB=20TO0STEP-1:POKE5,B:POKEW,129
2820 POKEA,15:POKEH,7:POKEL,12:NEXT
2830 POKEW,0:POKEA,0
2840 POKE53281,8:FORJ=1TO20:NEXT
2850 POKE53281,0
2860 RETURN
2870 REM----MINE/DETECTOR COLLISION----
2880 CODE=1024+INT((Y-50)/8+1)*40+INT((X-18)/8):REM CON
VERT X,Y TO SCREEN CODE
2890 IFPEEK(CODE)=32THEN3060
2900 IFPEEK(CODE+1)=32THENCODE=CODE-1
2910 REM---SOUND EFFECT---
2920 A=54277:S=54296:W=54276:H=54273:L=54272
2930 FORB=15TO0STEP-1
2940 POKE54272+CODE,B
2950 POKE54272+CODE+1,B:POKE53280,B+1
2960 POKE5,B:POKEW,129
2970 POKEA,15:POKEH,7:POKEL,12:NEXT
2980 POKEW,0:POKEA,0
2990 REM---TURN OFF MINE---
3000 POKECODE,32:POKECODE+1,32
3010 REM---RESET BORDER---
3020 POKE53280,14
3030 REM---SCORE---
3040 YS=YS+100
3050 PRINTTAB(6)CHR$(145)"YOUR SCORE";YS
3060 RETURN
3070 REM----OUT OF TIME ROUTINE----
3080 PRINTCHR$(147)
3090 PRINT:PRINT:PRINT:PRINT:PRINT
3100 PRINTTAB(12)CHR$(30)"  "
3110 PRINTTAB(12)"  "
3120 PRINTTAB(12)"  "
3130 PRINTTAB(12)"  "
3140 PRINTTAB(12)"  "
3150 PRINT:PRINT:PRINT
3160 PRINTTAB(13)"  "

```



```
3530 REM:
3540 REM-----DATA FOR MINES-----
3550 DATA0,0,3,63,122,63,15,0
3560 DATA0,0,192,252,94,252,240,0
```

Look Out!



A game for TWO players with joysticks. Firstly, set the target score – it may be the first to reach 10, 20 etc. – then each player uses his/her joystick to move around the screen. Don't cross the other player's track or turn back on yourself, or you lose!

```

1000 REM *** LOOK OUT! ***
1010 S=54272
1020 FOR I=0 TO 24:POKE S+I,0:NEXT
1030 POKES+24,14
1040 POKES+5,31:POKE S+6,15
1050 POKES,200
1060 PRINT CHR$(147)
1070 PRINT "INPUT SCORE TARGET: ";
1080 INPUT Q$:Q=VAL(Q$)
1090 IF Q<1 THEN 1060
1100 PRINT CHR$(147)
1110 POKE 53281,0:POKE 53280,0
1120 SC=1024:CO=55296:Z1=0:Z2=0
1130 X1=19:Y1=12:X2=21:Y2=12
1140 PRINT CHR$(19)CHR$(158)"PLAYER 1:"$1
1150 PRINT CHR$(19)TAB(24)"PLAYER 2:"$2
1160 REM ----- MAIN LOOP -----
1170 P1=PEEK(56320):P2=PEEK(56321)
1180 XO=X1:YO=Y1:OX=X2:OY=Y2
1190 IF P1=126 THEN Z1=1
1200 IF P2=254 THEN Z2=1
1210 IF P1=125 THEN Z1=2
1220 IF P2=253 THEN Z2=2
1230 IF P1=123 THEN Z1=3
1240 IF P2=251 THEN Z2=3
1250 IF P1=119 THEN Z1=4
1260 IF P2=247 THEN Z2=4
1270 IF RND(1)>.5 THEN 1330
1280 ON Z1 GOSUB 1450,1470,1490,1510
1290 ON Z2 GOSUB 1530,1550,1570,1590
1300 IF PEEK(SC+X1+40*Y1)=160 THEN 1620
1310 IF PEEK(SC+X2+40*Y2)=160 THEN 1670
1320 GOTO 1370
1330 ON Z2 GOSUB 1530,1550,1570,1590
1340 ON Z1 GOSUB 1450,1470,1490,1510
1350 IF PEEK(SC+X1+40*Y1)=160 THEN 1620
1360 IF PEEK(SC+X2+40*Y2)=160 THEN 1670

```

```

1370 POKESC+X0+40*Y0,160
1380 POKESC+0X+40*0Y,160
1390 POKECO+X1+40*Y1,14
1400 POKESC+X1+40*Y1,177
1410 POKECO+X2+40*Y2,5
1420 POKESC+X2+40*Y2,178
1430 GOTO1170
1440 REM ----- MOVEMENT -----
1450 Y1=Y1-1:IFY1<1THENY1=24
1460 RETURN
1470 Y1=Y1+1:IFY1>24THENY1=1
1480 RETURN
1490 X1=X1-1:IFX1<0THENX1=39
1500 RETURN
1510 X1=X1+1:IFX1>39THENX1=0
1520 RETURN
1530 Y2=Y2-1:IFY2<1THENY2=24
1540 RETURN
1550 Y2=Y2+1:IFY2>24THENY2=1
1560 RETURN
1570 X2=X2-1:IFX2<0THENX2=39
1580 RETURN
1590 X2=X2+1:IFX2>39THENX2=0
1600 RETURN
1610 REM ----- PLAYER 2 SCORES -----
1620 S2=S2+1:POKES+4,17
1630 POKES+1,30:FORT=1TO120:NEXT
1640 IFS2=0THEN1720
1650 POKES+4,0:GOTO1100
1660 REM ----- PLAYER 1 SCORES -----
1670 S1=S1+1:POKES+4,17
1680 POKES+1,30:FORT=1TO120:NEXT
1690 IFS1=0THEN1790
1700 POKES+4,0:GOTO1100
1710 REM ----- PLAYER 1 WINS -----
1720 PRINTCHR$(147):POKES+4,0

```

Listing continued next page

```
1730 PRINT"PLAYER 1 WINS THE GAME"
1740 PRINT:PRINT:PRINT"ANOTHER GO?"
1750 GETA$:IFA$<>"Y"AND A$<>"N"THEN1750
1760 IFA$="Y"THENRUN
1770 POKE53280,14:POKE53281,6:END
1780 REM ----- PLAYER 2 WINS -----
1790 PRINTCHR$(147):POKE53280,0
1800 PRINT"PLAYER 2 WINS THE GAME"
1810 PRINT:PRINT:PRINT"ANOTHER GO?"
1820 GETA$:IFA$<>"Y"AND A$<>"N"THEN1750
1830 IFA$="Y"THENRUN
1840 POKE53280,14:POKE53281,6:END
```

Drum Kit



The sound capabilities of the Commodore 64 are exceptional, but like most good things can be a pain to use. This drum simulation, however, gives a fair idea on how to obtain some percussive sounds.

The drums available are the BASS, SNARE, and HI and LO TOM-TOM's, and each different drum sound has been programmed in separate modules so that they can be utilized separately.

Even so, it is a complete program and on execution you can play each drum from the keyboard by pressing the keys 'F' and 'G' for Hi/Lo Tom-Toms, 'B' for the Bass Drum and 'J' for the Snare. Each drum's key has been positioned for easy access.

If a key is held down, that particular drum will keep on sounding and with a little bit of timing you will be able to play quite good rhythms.

The program has a lot of scope for expansion. Try adding a crash cymbal or hi-hat, for instance. Happy Drumming!

```

1000 REM ***: DRUM KIT ***
1010 PRINTCHR$(147)
1020 POKE53280,0:POKE53281,0
1030 PRINTTAB(4)"          "
1040 PRINTTAB(4)"          "
1050 PRINTTAB(4)"          "
1060 PRINTTAB(4)"          "
1070 PRINTTAB(4)"          "
1080 PRINT:PRINT:PRINT
1090 PRINTTAB(10)"[B] = BASS DRUM
1100 PRINTTAB(10)"[J] = SNARE DRUM
1110 PRINTTAB(10)"[F] = HI TOM
1120 PRINTTAB(10)"[G] = LO TOM
1130 PRINTTAB(10)"[ ] = CRASH CYMBAL
1140 PRINTTAB(10)"[R] = RANDOM DRUM SOLO
1150 PRINTTAB(10)"[S] = RETURN TO MANUAL
1160 REM ----- SET UP -----
1170 S=54272:S1=54286:S2=54279
1180 FORI=0TO24:POKEI+S,0:NEXT
1190 POKES+24,14:REM VOLUME
1200 REM ----- GET DRUMS -----
1210 GETA#
1220 IFA#="R"THEN1720:REM DRUM SOLO
1230 IFA#="J"THENGOSUB1300:REM SNARE
1240 IFA#="B"THENGOSUB1380:REM BASS
1250 IFA#="F"THENGOSUB1470:REM TOM 1
1260 IFA#="G"THENGOSUB1560:REM TOM 2
1270 IFA#=" "THENGOSUB1650:REM CYMBAL
1280 GOTO1210
1290 REM ----- SNARE DRUM -----
1300 POKES2+4,0
1310 POKES2+5,8
1320 POKES2+6,0
1330 POKES2+4,129
1340 POKES2,200
1350 POKES2+1,90
1360 RETURN

```



```

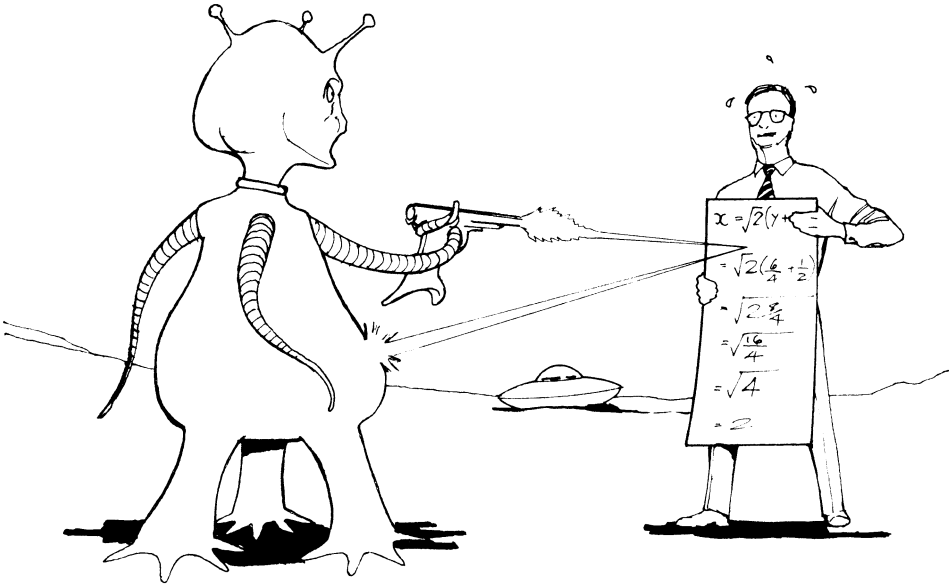
1370 REM ----- BASS DRUM -----
1380 POKES1+4,0
1390 POKES2+4,0
1400 POKES1+5,4
1410 POKES1+6,2
1420 POKES1+4,17
1430 POKES1,2
1440 POKES1+1,10
1450 RETURN
1460 REM ----- TOM 1 -----
1470 POKES1+4,0
1480 POKES2+4,0
1490 POKES1+5,4
1500 POKES1+6,2
1510 POKES1+4,17
1520 POKES1,2
1530 POKES1+1,20
1540 RETURN
1550 REM ----- TOM 2 -----
1560 POKES1+4,0
1570 POKES2+4,0
1580 POKES1+5,4
1590 POKES1+6,2
1600 POKES1+4,17
1610 POKES1,2
1620 POKES1+1,15
1630 RETURN
1640 REM ----- CRASH CYMBAL -----
1650 POKES+4,0
1660 POKES+5,9
1670 POKES+6,0
1680 POKES+4,129
1690 POKES,2
1700 POKES+1,150
1710 RETURN
1720 REM ---- RANDOM DRUM SOLO ----

```

Listing continued next page

```
1730 R=INT(RND(TI)*5+1)
1740 D=INT(RND(1)*100)
1750 FORT=1T050+D:NEXT
1760 ONRGOSUB1300,1380,1470,1560,1650
1770 GETB$:IFB$<>"S"THEN1730
1780 GOTO1210
```

Math Encounters



In this game your mathematical skill is called upon to save the humanoid. Answer the sum correctly and the ‘Force of Knowledge’ is with him. If you get the answer wrong you lay him open to liquidation by the alien.

The ‘+’ sign can be changed to ‘-’, ‘*’, or ‘/’ by altering lines 1720 and 1740. The range of the numbers can also be altered by changing lines 1690 and 1700.

This program demonstrates the lower case character set. Type it in as written and then press SHIFT and the Commodore Flag key to see lower case letters appear on the screen.

```

1000 REM *** MATH ENCOUNTERS ***
1010 PRINTCHR$(147)CHR$(14)
1020 POKE53280,0:POKE53281,0
1030 PRINTTAB(5)"|O PLAY \MATH ENCOUNTERS YOU"
1040 PRINTTAB(5)"MUST TYPE IN THE CORRECT"
1050 PRINTTAB(5)"ANSWER TO THE #UM AND PRESS"
1060 PRINTTAB(5)"THE _| _/ KEY."
1070 PRINT
1080 PRINTTAB(5)"|HEN WATCH THE #LIEN FIRE HIS"
1090 PRINTTAB(5)"LASER GUN."
1100 PRINTTAB(5)"\F THE WRONG ANSWER HAS BEEN"
1110 PRINTTAB(5)"GIVEN, THE POOR JUMANOID WILL"
1120 PRINTTAB(5)"PERISH..."
1130 PRINTTAB(5)"|UT IF THE _| || ANSWER WAS"
1140 PRINTTAB(5)"GIVEN THIS WILL GIVE THE"
1150 PRINTTAB(5)"JUMANOID A FORCE FIELD, TO"
1160 PRINTTAB(5)"WHICH THE LASER BOLT WILL"
1170 PRINTTAB(5)"REBOUND AND DESTROY THE ALIEN"
1180 PRINT
1190 PRINTTAB(5)"WHAT HAPPENS NEXT ??????"
1200 PRINTTAB(5)"DU GET ANOTHER #UM."
1210 PRINTTAB(5)"\S N'T I T G R E A T !!!"
1220 PRINT
1230 PRINTTAB(9)"_L_# # _| | _L_!"
1240 GETC#:IFC#=""THEN1240
1250 PRINTCHR$(147):POKE53280,4
1260 REM --- SET UP SPRITES & SOUND ---
1270 FORI=0TO62:READJ:POKEI+832,J:NEXT
1280 FORI=0TO62:READJ:POKEI+896,J:NEXT
1290 POKE981,3:REM BULLET SPRITE
1300 VC=53248:S=54272:S1=54286
1310 POKE2040,13:POKE2041,14:POKE2042,15
1320 POKEVC+21,0:POKE930,170
1330 POKEVC+28,3:POKEVC+37,10
1340 POKEVC+38,9:POKEVC+39,5
1350 POKEVC+40,5:POKEVC+41,1
1360 POKEVC,50:POKEVC+1,100

```

```

1370 POKEVC+2,255:POKEVC+3,100
1380 POKEVC+4,245:POKEVC+5,100
1390 POKEVC+23,7:POKEVC+29,7
1400 FORI=0TO24:POKEI+S,0:NEXT
1410 POKES+24,14
1420 POKES+5,31:POKES+6,15
1430 POKES1+5,31:POKES1+6,15
1440 POKES,200:POKES1,200
1450 REM ----- DRAW SCREEN -----
1460 PRINTCHR$(19)CHR$(158)
1470 FORT=1TO8:PRINT:NEXT
1480 FORT=1TO40:PRINT"_":NEXT
1490 PRINTCHR$(18);
1500 PRINTCHR$(156);
1510 FORT=1TO80:PRINT" ":NEXT
1520 PRINTCHR$(158)CHR$(146);
1530 FORT=1TO40:PRINT"~":NEXT
1540 FORT=1TO20:R=INT(RND(1)*360)
1550 POKE55296+R,7:POKE1024+R,46:NEXT
1560 PRINT:PRINTCHR$(30);
1570 PRINTTAB(11)"|ET THE SUM _ _|!!"
1580 PRINT
1590 PRINTTAB(10)"TO SAVE THE JUMANOID"
1600 PRINTCHR$(158)
1610 FORT=1TO40:PRINT"_":NEXT
1620 PRINTCHR$(19);
1630 PRINTCHR$(18)"          *** \MATH  ";
1640 PRINT"ENCOUNTERS ***          "
1650 REM ----- MAIN PROG -----
1660 FORT=1TO2000:NEXT:POKEVC+21,3
1670 B=245:D=-2
1680 FORI=1864TO1903:POKEI,32:NEXT
1690 X=INT(RND(TI)*20):REM CHANGE THESE
1700 Y=INT(RND(TI)*20):REM NO'S TO SUIT
1710 POKE781,21:POKE782,13:SYS65520
1720 PRINTX+"Y"="";

```

Listing continued next page

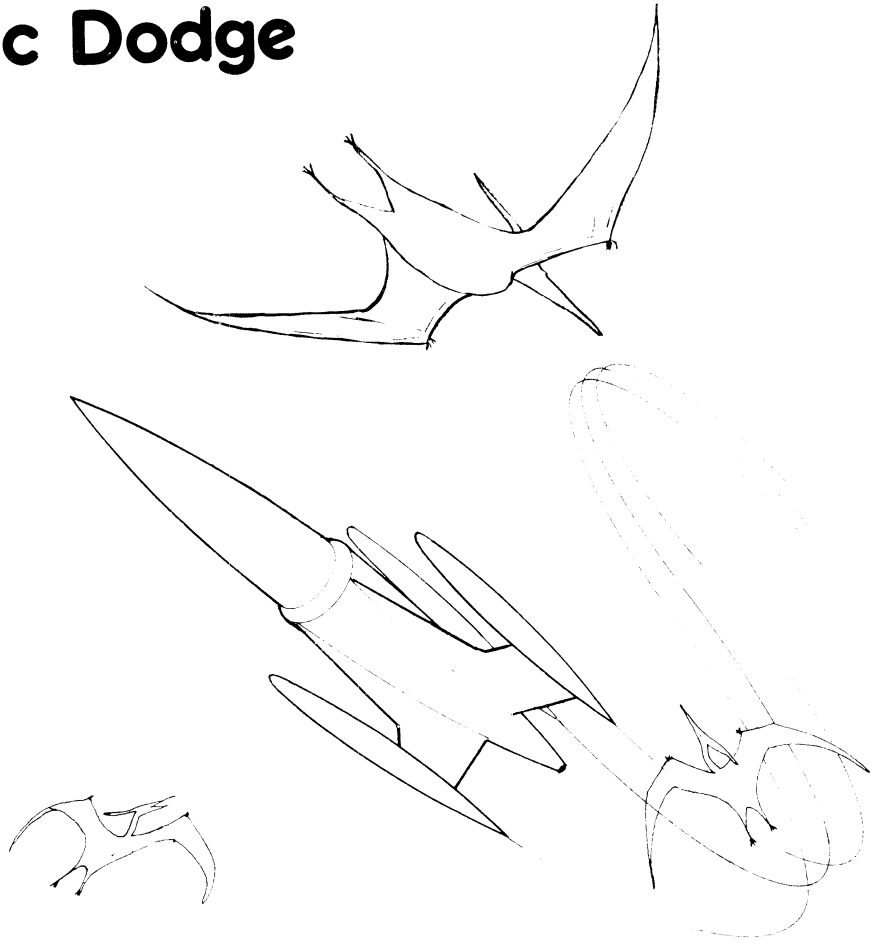
```

1730 INPUTQ$:Q=VAL(Q$):M=0
1740 IFQ<>(X+Y)THENM=1
1750 IFM=1THENGOSUB1790
1760 IFM=0THENGOSUB1920
1770 GOTO1660
1780 REM ----- WHOOPS! -----
1790 POKEVC+21,7
1800 GOSUB2040
1810 B=B-2:IFB>50THENPOKEVC+4,B:GOTO1810
1820 POKES+4,33:POKES1+4,33
1830 FORT=1T0100:POKEVC+21,3
1840 POKES+1,110-T:POKES1+1,101-T
1850 POKEVC+21,2:NEXTT
1860 POKES+4,0:POKES1+4,0
1870 POKE781,21:POKE782,7:SYS65520
1880 PRINT"|HE ANSWER TO THE SUM IS"X+Y
1890 FORD=1T08000:NEXT
1900 RETURN
1910 REM ----- HOORAY! -----
1920 POKEVC+21,7
1930 GOSUB2040
1940 B=B+D:IFB<84THEND=-D:POKE930,138
1950 POKEVC+4,B
1960 IFB<250THEN1940
1970 POKES+4,33:POKES1+4,33
1980 FORT=1T0100:POKEVC+21,3
1990 POKES+1,10+T:POKES1+1,1+T
2000 POKEVC+21,1:NEXT:POKE930,170
2010 POKES+4,0:POKES1+4,0
2020 RETURN
2030 REM ----- FIRE LASER -----
2040 POKES+4,33:POKES1+4,33
2050 FORG=40T010STEP-5
2060 POKES+1,G:POKES1+1,G+28
2070 NEXTG
2080 POKES+4,0:POKES1+4,0
2090 RETURN

```

```
2100 REM ----- SPRITE DATA -----
2110 DATA0,0,0,0,60,0,0,255,0,0,215,15
2120 DATA0,215,60,0,20,48,0,191,192,2
2130 DATA191,128,10,170,160,40,186,40
2140 DATA96,170,9,16,186,4,0,255,0,0
2150 DATA255,0,0,255,0,0,195,0,0,195,0
2160 DATA0,195,0,0,195,0,0,130,0,3,195,192
2170 DATA0,0,0,0,0,0,8,0,32,2,0,128,2
2180 DATA40,128,0,170,0,0,170,0,242,238
2190 DATA128,58,170,160,2,170,160,0,170
2200 DATA32,0,138,0,0,170,0,0,170,0,0
2210 DATA168,0,0,136,0,0,136,0,0,136,0
2220 DATA2,138,0,0,0,0,0,0,0,0
```

Cosmic Dodge



In Cosmic Dodge you are aboard a space freighter which has no forms of defence. You are heading straight into a swarm of bird-like creatures which will not attack you, but may very well bump into you. You have, however, two options:

- (1) Dodge them.
- (2) Use the hyperspace.

If you use the hyperspace option you do not score any points, and it could be disastrous if you come out of warp drive straight into one of the aliens, so remember where your ship was when you last saw it!

The F1 function key takes you into hyperwarp and F7 brings you back again.
Happy dodging!


```

1000 REM *** COSMIC DODGE ***
1010 PRINTCHR$(147):GOSUB1700:REM TITLE
1020 POKE53280,0:POKE53281,0
1030 REM *** READ JOYSTICK DATA ***
1040 FORI=49183TO49478:READJ
1050 POKEI,J:CC=CC+J:NEXTI:READJ
1060 IFJ<>CCTHENPRINT"DATA ERROR":END
1070 SYS49183
1080 REM *** READ SPRITE DATA ***
1090 FORI=0TO62:READJ:POKE832+I,J:NEXT
1100 FORI=0TO62:READJ:POKE896+I,J:NEXT
1110 REM *** SET JOYSTICK PARAMETERS ***
1120 POKE49152,50:POKE49153,0
1130 POKE49156,255:POKE49157,0
1140 POKE49160,180:POKE49161,0
1150 POKE49164,207:POKE49165,0
1160 POKE49168,0
1170 REM *** SET UP SPRITES ***
1180 POKE2040,13:VC=53248
1190 POKEVC+39,0:POKEVC+31,0
1200 POKEVC+23,1:POKEVC+29,1
1210 X=170:Y=207:POKEVC+1,Y:POKEVC,X
1220 POKE49169,X:POKE49173,Y
1230 S=54272
1240 FORI=0TO24:POKES+I,0:NEXT
1250 POKES+5,31:POKES+6,240
1260 PRINTCHR$(17)CHR$(17)
1270 PRINTTAB(10)"PRESS A KEY TO PLAY"
1280 GETJ$:IFJ$<>" "THEN1280
1290 GETA$:IFA$=""THEN1290
1300 PRINTCHR$(147):POKEVC+21,1
1310 REM *** MAIN LOOP ***
1320 X=PEEK(49169):Y=PEEK(49173)
1330 PRINTCHR$(19)CHR$(17);
1340 PRINTCHR$(157)CHR$(148)
1350 POKE218,160

```

Listing continued next page

```

1360 POKE646,INT(RND(1)*14+1)
1370 PRINTTAB(INT(RND(TI)*35))" 4"
1380 PRINTCHR$(19)CHR$(158)"SCORE:"SE
1390 IFSE>HITHENHI=SE
1400 PRINTCHR$(19)TAB(23)"HIGH SCORE:"HI
1410 GETA$:IFA$=CHR$(133)THENGOSUB1500
1420 IFA$=CHR$(136)THENGOSUB1550
1430 IFHY=1THEN1330
1440 POKEVC+31,0
1450 IFPEEK(VC+31)AND1=1THEN1590
1460 IFHY=0THENSE=SE+1
1470 POKEVC,X:POKEVC+1,Y
1480 GOTO1320
1490 REM *** HYPER SPACE ***
1500 PRINTCHR$(30)
1510 PRINTCHR$(19)TAB(11)"-HYPERWARP-"
1520 POKEVC+21,0:HY=1
1530 RETURN
1540 REM *** WARP SHUT DOWN ***
1550 PRINTCHR$(19)TAB(11)"          "
1560 POKEVC+21,1:HY=0
1570 RETURN
1580 REM *** WHOOPS! ***
1590 POKE2040,14
1600 FORV=15TO0STEP-.3:POKEVC+21,1
1610 POKES+4,129:POKES,100:POKES+1,3+V
1620 POKES+24,V:POKEVC+21,0:NEXT
1630 FORK=1TO1000:NEXT
1640 POKE2040,13
1650 PRINTCHR$(19):FORT=1TO4:PRINTCHR$(17)
1660 NEXT
1670 PRINTTAB(14)"-GAME OVER-"CHR$(17)
1680 FORK=1TO1000:NEXT
1690 SE=0:GOSUB1700:GOTO1180
1700 REM *** TITLE PAGE ***
1710 PRINTCHR$(147)CHR$(158)
1720 PRINTCHR$(17)

```

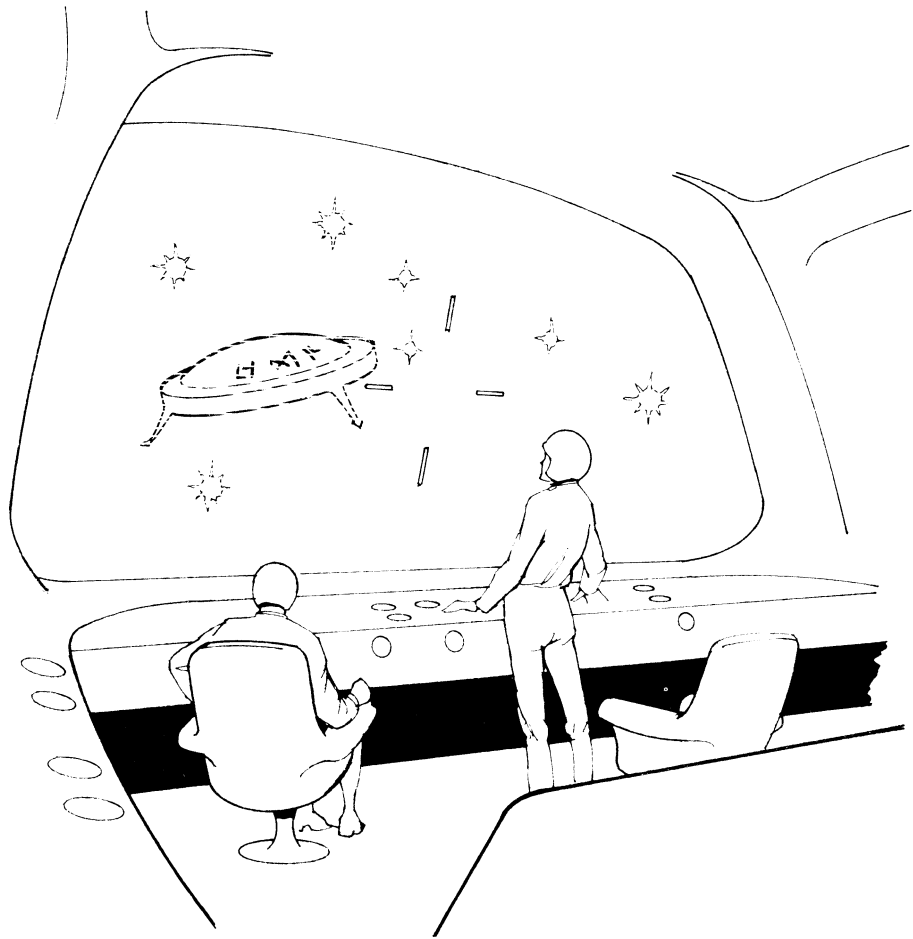


```

2090 DATA192,208,3,254,22,192,189,0,192
2100 DATA221,17,192,189,1,192,253,18
2110 DATA192,144,18,189,0,192,157,17
2120 DATA192,189,1,192,157,18,192,104
2130 DATA74,72,76,1,193,104,74,72,176
2140 DATA13,222,17,192,189,17,192,201
2150 DATA255,208,3,222,18,192,189,17
2160 DATA192,221,4,192,189,18,192,253,5
2170 DATA192,144,18,189,4,192,157,17
2180 DATA192,189,5,192,157,18,192,104
2190 DATA74,72,76,46,193,104,74,72,176
2200 DATA8,254,17,192,208,3,254,18,192
2210 DATA104,74,176,5,169,1,153,25,192
2220 DATA200,232,232,224,4,240,3,76,91
2230 DATA192,238,27,192,108,29,192
2240 DATA35994:REM*CHECKSUM*
2250 REM *** SPRITE 0 (SHIP) ***
2260 DATA0,32,0,0,32,0,0,32,0,0,112,0,0
2270 DATA80,0,0,112,0,0,112,0,0,112,0,0
2280 DATA248,0,0,248,0,0,248,0,0,248,0
2290 DATA0,248,0,2,250,0,2,170,0,6,171
2300 DATA0,6,171,0,6,171,0,6,171,0,7,39
2310 DATA0,7,39,0
2320 REM *** SPRITE 0 (EXPL) ***
2330 DATA0,16,64,34,16,128,19,9,16,8
2340 DATA200,96,4,68,64,2,32,128,48,9,0
2350 DATA12,172,28,2,68,224,0,17,0,1
2360 DATA168,48,24,37,8,97,64,128,2,8
2370 DATA96,4,136,16,1,36,140,2,36,64,4
2380 DATA68,32,8,66,0,16,2,0,0,0,0

```

Galactic Battle



Just a little game for 'Shoot 'em up' lovers, Galactic Battle takes place in a far-away galaxy. You are in charge of a scout ship which was part of a convoy. The rest of the convoy, however, was destroyed by an alien force. You are all alone. Your sub-space radio is out of action and you have only a small amount of energy left. Then, just when you thought you had escaped the clutches of the evil alien force you are again under attack. With only a limited amount of energy how long can you last?

RUNning this game puts you in the cockpit of the scout ship, looking out into the void of space. Below the window is the console which keeps you updated on your ship's status. In the centre of the console, ticking away, is a clock to show how long you last. In the middle of the window you will see a cross-hair sight into which you must manouevre the alien in order to destroy it. When the joystick is being moved left, right, up or down the fire button is inoperable to stop itchy fingers holding the button down all the time. If you run out of energy your shields will be no more and as soon as the damage report tops 500 . . . Complete Destruction. At this point you will be given the option to prove you are not a chicken!

Here is a list of VARIABLES used in this program:

- SC = Screen base location
- CO = Color base location
- SE = Screen explosion location
- CE = Color explosion location
- X = Left and right movement of alien
- Y = Up and down movement of alien
- A() = Array holding direction of debris
- EN = Current energy supply
- KI = Number of aliens destroyed
- DA = Current damage report
- PH = Photons left
- S = Start of 'SID' chip
- VC = Start of 'VIC' chip

One word of warning. SAVE this program before RUNning it because it contains a SYS64738 which will erase the program. This is used if the player decides not to play another game. To avoid this, simply press RUN/STOP and RESTORE when you are asked to play again.

```

1000 REM***                      ***
1010 REM -("<img alt="three vertical bars" data-bbox="315 288 345 303"/>"=SYMBOL FOR CURSOR LEFT)-
1020 :
1030 PRINTCHR$(147)CHR$(152)
1040 POKE53280,11:POKE53281,0
1050 :
1060 REM ----- READ SPRITE DATA -----
1070 :
1080 FORI=0TO62:READJ:POKEI+832,J:NEXT
1090 FORI=0TO62:READJ:POKEI+896,J:NEXT
1100 :
1110 REM ----- SET UP SPRITES -----
1120 :
1130 VC=53248:POKEVC+21,3
1140 POKE2040,13:POKE2041,14
1150 POKEVC+23,0:POKEVC+29,3
1160 POKEVC+39,7:POKEVC+40,12
1170 POKEVC,161:POKEVC+1,107
1180 :
1190 REM ----- SET UP VARIABLES -----
1200 :
1210 SC=1024:CO=55296
1220 SE=SC+380:CE=CO+380
1230 X=INT(RND(1)*200+30)
1240 Y=INT(RND(1)*150+10)
1250 A(1)=1:A(2)=39:A(3)=40:A(4)=41
1260 A(5)=-1:A(6)=-39:A(7)=-40:A(8)=-41
1270 EN=999:KI=0:DA=0:PH=99
1280 TI$="000000"
1290 :
1300 REM --- SET UP SID PARAMETERS ---
1310 :
1320 S=54272
1330 FORI=0TO24:POKES+I,0:NEXT
1340 POKES+24,14
1350 POKES+5,31:POKES+6,15

```

Listing continued next page

```

1360 POKES,150:POKES+1,1
1370 :
1380 REM ----- SET UP SHIPS CONSOLE -----
1390 :
1400 PRINTCHR$(151)
1410 POKE781,17:POKE782,0:SYS65520
1420 PRINTCHR$(18)"▀ ";
1430 FORT=1TO36:PRINT"__";:NEXT
1440 PRINT" ▀";
1450 PRINT" | _____ ";
1460 PRINT" | _____ | ";
1470 PRINT" | ENERGY: | _____ ";
1480 PRINT"_____| PHOTON: | | ";
1490 PRINT" | _____ | ";
1500 PRINT" | | _____ | ";
1510 PRINT" | DAMAGE: | _____ ";
1520 PRINT"_____| KILLS : | | ";
1530 PRINT" | _____ "
1540 PRINT" | _____ | ";
1550 PRINT" L";
1560 FORT=1TO36:PRINT"__";:NEXT
1570 PRINT"└ ";CHR$(158);
1580 FORI=0TO39:POKE1984+I,160
1590 POKE56256+I,11:NEXT
1600 FORT=1TO25:Q=INT(RND(1)*680)
1610 POKEQ+Q,7:POKESC+Q,46:NEXT
1620 :
1630 REM ----- MAIN LOOP -----
1640 :
1650 P=PEEK(56320):Z=0:R=0
1660 IFF=126THENZ=1
1670 IFF=125THENZ=2
1680 IFF=123THENZ=4
1690 IFF=119THENZ=3
1700 IFF=111THENPH=PH-1:GOSUB2380
1710 ONZGOSUB1950,1970,1990,2010
1720 POKEVC+30,0

```



```

1730 IFP=111AND(PEEK(VC+30)AND3)=3THENGOSUB2150
1740 EN=EN-.1:IFEN<0THEN2430
1750 IFRND(1)<.025THENGOSUB2060
1760 IFRND(1)<.2THENR=INT(RND(TI)*4+1)
1770 ONRGOSUB1950,1970,1990,2010
1780 IFDA>500THEN2430
1790 POKE781,19:POKE782,11:SYS65520
1800 PRINT;INT(EN);"III "
1810 POKE781,21:POKE782,11:SYS65520
1820 PRINT;DA;
1830 POKE781,19:POKE782,32:SYS65520
1840 PRINT;PH;"III "
1850 POKE781,21:POKE782,32:SYS65520
1860 PRINT;KI
1870 POKE781,20:POKE782,18:SYS65520
1880 PRINTMID$(TI$,3,1)MID$(TI$,4,1);
1890 PRINT":":RIGHT$(TI$,2)
1900 POKEVC+2,X:POKEVC+3,Y
1910 GOTO1650
1920 :
1930 REM ----- ALIEN MOVEMENT -----
1940 :
1950 Y=Y-8:IFY<10THENY=10
1960 RETURN
1970 Y=Y+8:IFY>165THENY=165
1980 RETURN
1990 X=X-8:IFX<10THENX=10
2000 RETURN
2010 X=X+8:IFX>246THENX=246
2020 RETURN
2030 :
2040 REM ----- ALIEN RETURNS FIRE -----
2050 :
2060 POKE53281,2:POKEVC+40,1
2070 POKES+4,129:DA=DA+INT(RND(1)*30)
2080 EN=EN-10:IFEN<0THEN2430

```

Listing continued next page

```

2090 FORD=1T0100:NEXT:POKEVC+40,12
2100 POKES+4,0:POKE53281,0
2110 RETURN
2120 :
2130 REM ----- ALIEN  EXPLODES -----
2140 :
2150 POKEVC+21,1:POKES+4,129
2160 POKE53281,7:POKE53280,12
2170 FORD=1T050:NEXTD:POKE53281,0
2180 POKE53280,11
2190 FORT=1T07:FORD=1T025:NEXTD
2200 G=INT(RND(TI)*8+1):W(T)=G
2210 POKECE+T*A(G),1:POKESE+T*A(G),127
2220 NEXTT:KI=KI+1
2230 FORT=1T07STEP.5
2240 POKESE+T*A(W(T)),32
2250 NEXT:POKES+4,0
2260 :
-2270 REM ----- NEXT ALIEN -----
2280 :
2290 FORJ=1T02000:NEXT
2300 X=INT(RND(1)*200+30)
2310 Y=INT(RND(1)*150+10)
2320 POKEVC+2,X:POKEVC+3,Y
2330 POKEVC+21,3
2340 RETURN
2350 :
2360 REM ----- CHECK  PHOTONS -----
2370 :
2380 IFPH>0THENRETURN
2390 PH=99:EN=EN-200:RETURN
2400 :
2410 REM ----- INTO OBLIVION -----
2420 :
2430 POKEVC+21,0:FORI=15T00STEP-1
2440 FORC=1T015
2450 POKE53281,C:POKE53280,C

```

```

2460 POKES+4,129:POKES+1,20+I
2470 POKES+24,I:NEXTC,I
2480 POKES+4,0:POKE53281,0:POKE53280,0
2490 PRINTCHR$(158)CHR$(147)
2500 PRINT"ANOTHER FIGHT ";
2510 PRINT"OR ARE YOU CHICKEN?"
2520 PRINT:PRINT:PRINT
2530 PRINT"YES OR NO?....."
2540 GETA$:IFA$<>"Y"AND A$<>"N"THEN2540
2550 IFA$="Y"THENRUN
2560 SYS64738:REM WARNING CLEARS MEMORY
2570 :
2580 REM ----- SPRITE DATA -----
2590 :
2600 DATA0,16,0,0,16,0,0,16,0,0,16,0,0
2610 DATA0,0,0,0,0,0,0,0,0,0,0,0,0
2620 DATA0,0,254,0,254,0,0,0,0,0,0,0,0
2630 DATA0,0,0,0,0,0,0,0,0,0,0,16,0
2640 DATA16,0,0,16,0,0,16,0
2650 :
2660 DATA0,0,0,0,0,0,0,0,0,0,0,0,0
2670 DATA0,0,0,0,120,0,7,207,128,60
2680 DATA120,240,96,0,24,71,255,136
2690 DATA124,120,248,88,48,104,76,48
2700 DATA200,71,255,136,64,0,8,224,0
2710 DATA28,160,0,20,160,0,20,224,0,28
2720 DATA0,0,0
2730 :
2740 REM ----- END OF PROGRAM -----

```


Terry Peter Barrett is a freelance programmer. He formerly worked on electronic hardware with Plessey Telecommunications.

Stephen William Colwill is Head of Computer Studies at Sandhurst School. Prior to this he was Assistant Teacher of Mathematics, Computer Studies and Statistics at Altwood School in Maidenhead (1981-1983). He holds a B.Eng. (Hons.) in Civil Engineering from the University of Liverpool (1980) and a Cert. Education from the University of Reading (1981).

Move into the fast lane with twenty sophisticated, skill-building game programs for your Commodore.

WINNING GAMES on the **COMMODORE 64**

Terry Barrett and Stephen Colwill

If you're a beginner looking for sound programming advice . . . a quick learner ready for more advanced techniques . . . a seasoned programmer who wants to write commercial-quality arcade game programs . . . then this book is for you.

Winning Games on the Commodore 64 is a totally enjoyable guide that lets you mix the business of learning with the pleasure of game playing. Here are twenty high-level, ready-to-run arcade game programs for the Commodore — games that build your programming skills as you speed up the action to more challenging levels of play. Each game is accompanied by a brief explanation of how it was created and how to get the most out of it.

You'll master puzzlers such as *Math Encounter*, a program that calls upon your math skills to save the "humanoid". One wrong answer can lay him open for liquidation by an alien. Another game, *Laser Spell*, requires you to use your joystick to hit words with laser fire.

As you code in the games, you'll learn basic programming techniques, plus hints on writing faster moving graphics, basic loaders for machine code routines, and tips on constructing your own user characters. And for those interested in the "hows" and "whys" of high-level game programming, one program is thoroughly dissected and analyzed, so you can see and understand its structure and the techniques used to create it.

Terry Barrett is a freelance programmer.

Stephen Colwill is Head of Computer Studies, Sandhurst School, Surrey, England.

Wiley Press Guides have taught more than three million people to use, program, and enjoy microcomputers. Look for them all at your favorite bookshop or computer store.

Recreational Computing Series from
WILEY PRESS

A Division of John Wiley and Sons, Inc.

605 Third Avenue, New York, NY 10158

New York · Chichester · Brisbane · Toronto · Singapore

ISBN 0-471-87335-7